Introductie Domein Decompositie en Parallel Rekenen in SIMONA

Cursusboek Module 1

Rijkswaterstaat
Ministerie van Infrastructuur en Waterstaat

Introductie Domein Decompositie en Parallel Rekenen in SIMONA

Cursusboek Module 1

Version : 1.0, oktober 2002
Maintenance : see www.helpdeskwater.nl/waqua

Inhoud

1 Introductie

1.1 Parallel Rekenen en Domein Decompositie

Het principe achter parallel rekenen is eenvoudig. Stel dat men een woonwijk met allemaal identieke huizen wil bouwen. Dan kan men één groep bouwvakkers aan het werk zetten die achtereenvolgens alle huizen bouwen. Maar het gaat uiteraard sneller als men meerdere groepen bouwvakkers inzet die ieder een aantal huizen bouwen.

Zo ook kan men een stuk rekenwerk door één computer laten uitvoeren, maar het gaat sneller als men het rekenwerk verdeelt over meerdere computers of processoren. De hoeveelheid werk en de kosten blijven hetzelfde, maar het duurt minder lang om het uit te voeren.

Parallel rekenen wordt in SIMONA gebruikt om de doorlooptijd van simulaties te bekorten. Het rekenrooster wordt in stukken verdeeld en elk van de stukken wordt op een aparte processor doorgerekend. De stukken van het rekenrooster zijn dan de ‘huizen’ in het bovenstaande voorbeeld en processoren zijn de groepen bouwvakkers.

In het voorbeeld boven wordt een woonwijk gebouwd met identieke huizen. Veel woonwijken zullen echter bestaan uit verschillende delen met andere huizen. De woonwijk is als het ware verdeeld (gedecomponeerd) in verschillende domeinen en per domein wordt een speciaal soort huis gebouwd. Merk op dat de term Domein Decompositie in dit voorbeeld dus vooral gaat over het ontwerp van de woonwijk en niet primair over de manier waarop de woonwijk daadwerkelijk door bouwvakkers gebouwd wordt.

In SIMONA betekent Domein Decompositie dat er niet langer gewerkt wordt met één enkel rekenrooster maar dat het door te rekenen gebied wordt verdeeld in een aantal domeinen en per domein wordt een apart rooster gebruikt. De roosters voor de verschillende domeinen kunnen verschillen qua verticale fijnheid (d.i. het aantal lagen) of qua horizontale fijnheid (d.i. roosterafstand).

De verschillende domeinen kunnen op verschillende processoren worden doorgerekend, maar dat hoeft niet. Het is ook mogelijk om deze domeinen op hun beurt weer parallel door te rekenen. Parallel rekenen en domein decompositie zijn dus wezenlijk onafhankelijke technieken. Wel is het zo dat domein decompositie in SIMONA is geïmplementeerd als generalisatie van parallel rekenen.

1.2 Geschiedenis van parallel rekenen en domein decompositie voor SIMONA

De ontwikkeling van parallel WAQUA/TRIWAQ is tien jaar geleden gestart. In 1992 ondersteunde de dienst getijdenwateren (voorloper van RIKZ) het promotieonderzoek van Edwin Vollebregt en Mark Roest aan de Technische Universiteit Delft, dat mede tot doel had om een parallelle versie van TRIWAQ te ontwikkelen.

In 1996 werd duidelijk dat de parallellisatie succesvol was: experimenten op een cluster van HP werkstations en op een Cray T3D toonden een goede versnelling van de simulaties. Bovendien werd duidelijk dat de gekozen aanpak ook geschikt zou kunnen zijn voor de implementatie van domein decompositie. Daarom werd besloten om de ontwikkeling van parallel WAQUA/TRI-WAQ door te zetten.

De twee promovendi richtten een eigen bedrijf, om de opgedane kennis verder uit te bouwen en beschikbaar te maken voor bedrijven. De eerste projecten van dit nieuwe bedrijf omvatten de integratie van de testversie van parallel TRIWAQ met de moederversie en een tweetal voorstudies naar de haalbaarheid van de parallellisatie van WAQUA. Om de schaalbaarheid naar hoge aanallen processen te onderzoeken werd parallel TRIWAQ ondermeer geïnstalleerd op een Cray T3E machine en op een IBM SP-2. Experimenten hiermee wezen uit dat voor grote modellen 48 processoren van de Cray T3E nuttig ingezet konden worden, waarmee simulaties met een factor 60 versneld konden worden ten opzichte van een enkel werkstation.

Tegelijkertijd werd ook de ontwikkeling van domein decompositie ter hand genomen. Bij het Waterloopkundig Laboratorium was men inmiddels al flink gevorderd met de ontwikkeling van domein decompositie, waarbij een iets andere aanpak gevolgd werd dan wat het uitbouwen van de parallelle versie zou opleveren. De vraag deed zich dan ook voor of de aanpak van het Waterloopkundig Laboratorium niet de voorkeur verdiende. Er werden enkele studies verricht om na te gaan hoe de bij RIKZ gekozen benadering zich verhield tot die van het Waterloopkundig Laboratorium. Het bleek dat de benaderingen conceptueel niet veel verschilden.

In 1998 werd een plan van aanpak opgesteld voor het realiseren van domein decompositie voor SIMONA en werd de eerste stap daarvoor gezet met het bouwen van een testversie van TRIWAQ waarin domein decompositie met verticale verfijning werd ondersteund. In het jaar daarop werd een testversie met horizontale verfijning ontwikkeld.

Inmiddels werd er gewerkt aan het beschikbaar maken van de parallelle functionaliteit voor eindgebruikers. In 1999 werd de parallelle functionaliteit geïntegreerd met de moederversie. Vanwege de enorme veranderingen die dit met zich meebracht in de moederversie kwam deze pas in de loop van 2000 beschikbaar voor eindgebruikers. Niettemin is al vanaf 1999 voor heel grote simulaties gebruik gemaakt van de nieuwe versie op de UNITE supercomputer van het academisch rekencentrum SARA in Amsterdam.

Eind 2000 werd begonnen met het integreren van de functionaliteit voor domein decompositie met verticale verfijning in de moederversie. In de export 2002-01 van SIMONA zijn deze aanpassingen officieel doorgevoerd.

Op dit moment worden de uitbreidingen voor domein decompositie met horizontale verfijning opgenomen in de moederversie van WAQUA/TRIWAQ in SIMONA. Daarna zullen uitbreidingen worden gemaakt waardoor horizontale en verticale verfijning samen in een enkele run kunnen worden gebruikt. Daarbij zullen ook de informaticatechnische verschillen tussen domein decompositie met horizontale en verticale verfijning worden gladgestreken. Ten slotte wordt nagedacht over de mogelijke verdere ontwikkeling van domein decompositie: verschillende tijdstappen per domein, verschillende fysische processen per domein, koppeling van WAQUA aan TRIWAQ, en koppeling van x-roosterlijnen aan y-roosterlijnen.

1.3 Huidig Gebruik van Parallel Rekenen en Domein Decompositie

Voor grootschalige simulaties maakt Rijkswaterstaat op dit moment gebruik van een drietal rekenvoorzieningen: de UNITE en de TERAS bij SARA en een cluster van 16 Linux PC’s dat is geïnstalleerd bij de Technische Universiteit Delft. Daarnaast hebben enkele directies van Rijkswaterstaat en ingenieursbureau’s die veel werk voor Rijkswaterstaat doen eigen rekenfaciliteiten aangeschaft, waaronder zowel multiprocessor werkstations als clusters van MS Windows of Linux PC’s.

Zowel de parallelle functionaliteit als domein decompositie met verticale verfijning worden intensief gebruikt voor grootschalige studies. In deze paragraaf zal een tweetal typische voorbeelden kort besproken worden.

1.3.1 Parallel rekenen met het Maasmodel

Ten behoeve van het randvoorwaardenboek 2001 is een WAQUA-model van de gehele Maas gemaakt. Dit model loopt van Eijsden

(km 2) tot aan Keizersveer (km 247). Het rooster van het model bestaat uit 107 x 4182 punten waarvan er ongeveer 210.000 actief zijn (47%). Voor berekeningen van hoogwatergolven worden typisch simulaties uitgevoerd van 10-15 dagen: een aanloopperiode van 6-9 dagen waarin de golf zich opbouwt, gevolgd door de looptijd van de piek van de golf van Eijsden naar Keizersveer. De gebruikte tijdstap is 15 seconden (55-90.000 tijdstappen).

In 1999 zijn de hydraulische ruwheden van het model afgeregeld om een zo goed mogelijke overeenstemming te bereiken tussen berekende waardes en een bepaalde set van gemeten waterstanden. Dit gebeurt via een algoritme dat verschillen tussen berekende en gemeten waterstanden vertaalt naar aanpassingen aan de ruwheden. Dit gebeurt in een aantal stappen, gaandeweg wordt een optimale instelling van de ruwheden verkregen. Vanwege het grote aantal vrijheidsgraden in het Maasmodel zijn er voor de inregeling ongeveer 50 sommen uitgevoerd.

Het Maasmodel is ook een belangrijk hulpmiddel voor de projectorganisatie De Maaswerken die allerlei mogelijke ingrepen in de Maas beoordeelt en uitvoert om de overstromingskans langs de rivier tot aanvaardbare niveaus terug te brengen. Ten eerste zijn er hiervoor referentieruns uitgevoerd met het oorspronkelijke Maasmodel voor verschillende scenario's. Een zo'n scenario is bijvoorbeeld een afvoergolf die met een kans van 1/250 per jaar optreedt. Vervolgens zijn simulaties uitgevoerd met diverse aangepaste versies van het WAQUA-model van de Maas. Bijvoorbeeld met op bepaalde plaatsen verhoogde kades, op andere plaatsen een verbreed en/of verdiept zomerbed, enz. In totaal zijn er de afgelopen drie jaar rond de 450 van dit soort simulaties uitgevoerd.

In 1999 is de rekentijd van het Maasmodel is onderzocht bij installatie van parallel WAQUA op de supercomputer UNITE van SARA. Op een enkele processor van deze machine kostte een simulatie van 4 uur werkelijke tijd zo'n 1,75 uur rekentijd, ofwel 4,4 dagen rekentijd per simulatie van 10 dagen. In parallelle runs op 16 processoren van de UNITE werd een versnelling met een factor 22,4 bereikt. Hiermee kan dezelfde simulatie van 10 dagen worden uitgevoerd in minder dan 5 uur. Dit voorbeeld toont aan dat parallel rekenen van cruciaal belang is voor een goede voortgang van dit soort projecten.

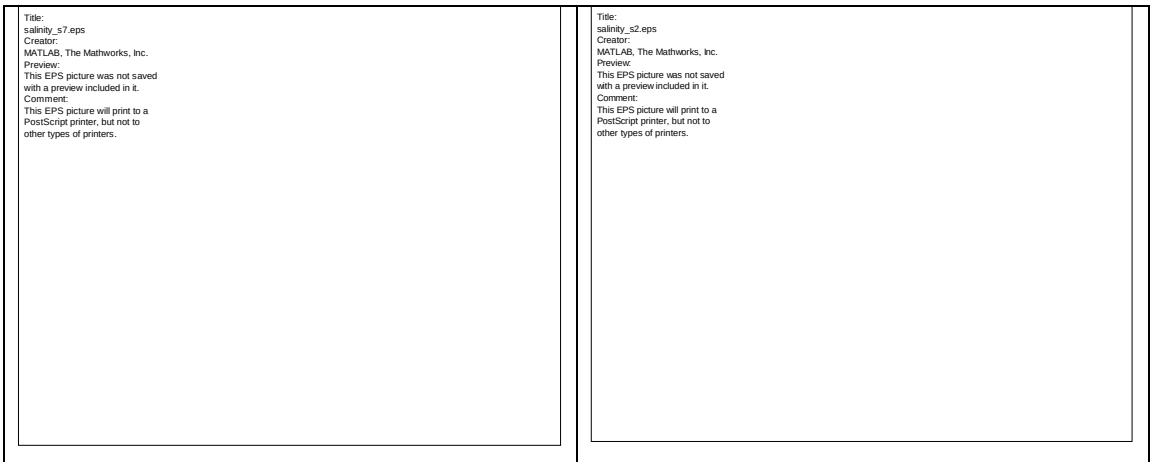
1.3.2 Domein decompositie voor het Rijn-Maasmond model

Het Rijn-Maasmond (RYMAMO) model is een model van de monding van de Maas, inclusief de Rotterdamse haven en een stuk van de open zee. Met dit model zijn uitvoerige testen gedaan kort

Figuur 1: de verdeling van het RYMAMO model in deeldomeinen

Figuur 2: de belangrijkste meetstations in het Rijnmond gebied

na het gereedkomen van de eerste operationele versie van TRIWAQ waarin domein decompositie met verticale verfijning wordt ondersteund. In deze testen is het model in drie stukken verdeeld (zie Figuur 1): de open zee (4 lagen), het Rijnmond



Figuur 3: zoutheid op 22 oktober 1985 om 12:00 in Secties 2 en 7

gebied (10 lagen) en de Rotterdamse haven en rivieren (2 lagen). Deze verdeling is gekozen omdat in het Maasmond gebied sterkte driedimensionale effecten (zoals gelaagdheid in zoutconcentraties) optreden, zodat men daar een relatief groot aantal lagen wil hebben. Het voordeel van gebruik van domein decompositie is dat men nu niet overal 10 lagen hoeft te gebruiken, waarmee de rekentijd voor twee dagen simulatietijd gereduceerd wordt van 11 uur tot 4.6 uur.

Belangrijk is uiteraard dat de resultaten van de simulatie niet significant worden aangetast. De effecten worden getoond in Figuur 3 voor een tweetal meetstations (zie ook Figuur 2). Sectie 2 bevindt zich in het gebied dat met 10 lagen wordt doorgerekend. Er zijn kleine verschillen te zien op kleine diepte, maar over het geheel genomen volgt de domeindecompositie versie goed de versie waarin overal 10 lagen zijn gebruikt. Sectie 7 bevindt zich in het gebied waarin slechts 2 lagen zijn gebruikt. Omdat er maar twee lagen zijn, kan het zoutheidsprofiel niet goed gevolgd worden. Niettemin is de voorspelling kwantitatief in de goede orde.

1.4 Toekomstige Ontwikkelingen

Op dit moment wordt ondermeer gewerkt aan het parallelliseren van het Kalman filter dat beschikbaar is in de Nautilus programmatuur (dit is een versie van WAQUA met data assimilatie functionaliteit). Een experimentele versie hiervan bestaat al en wordt voor onderzoek gebruikt op de TERAS machine van SARA en op het Linux cluster bij de TU Delft. Aan de hand van de experimetele versie wordt gewerkt aan het

parallelliseren van het Kalaman filter in de officiële Nautilus programmatuur.

Er wordt ook nagedacht over de verdere uitbouw van de domein decompositiefunctionaliteit, zoals,

- het gebruik van verschillende rekenmodellen (bijv. Wel of geen transport) in verschillende domeinen,
- verschillende tijdstappen voor de verschillende domeinen en
- niet-aansluitende roosters.

De ontwikkeling van het Open Model Systeem (OMS), waarbij SIMONA wordt samengevoegd met het Delft3D van WL, betekent dat ook de implementatie van de parallelle en domein decompositie functionaliteiten opnieuw bezien wordt. De verwachting is dat de implementatie van deze functionaliteit in OMS sterk zal lijken op die in SIMONA.

2 Resumé WAQUA/TRIWAQ

In dit hoofdstuk wordt een beknopte introductie gegeven van de simulatiemodellen WAQUA en TRIWAQ. Daarbij wordt vooral ingegaan op aspecten die later terugkomen bij de beschrijving van parallel rekenen en domein decompositie. Voor een uitgebreide introductie en beschrijving van WAQUA en TRIWAQ wordt verwezen naar de “Gebruikershandleiding WAQUA – Algemene Informatie”.

WAQUA is zowel de naam van het simulatiesysteem waarmee twee- en driedimensionale waterbewegings- en waterkwaliteitsberekeningen kunnen worden uitgevoerd, als de naam van het simulatiemodel voor de tweedimensionale (dieptegemiddelde) stromingsvergelijkingen. TRIWAQ is het driedimensionale model voor ondiepwatersimulatie. Het systeem stelt de gebruiker in staat tot simulatie van zowel stationaire- als niet-stationaire stromingssituaties, en van transport van opgeloste stoffen.

Het simulatiesysteem WAQUA is gebaseerd op SIMONA, een flexibel concept en bijbehorende programmatuuromgeving voor de ontwikkeling van modelprogrammatuur.

2.1 Introductie in ondiepwatersimulatie

Het wiskundige model voor de beweging van water is al lange tijd bekend. De zogenaamde Navier-Stokes vergelijkingen geven een goede beschrijving. Voor lange golven in kustwateren en rivieren kan een vereenvoudiging worden aangebracht, omdat de stroming voornamelijk in het horizontale vlak plaatsvindt. Bij dit soort stromingen is de verticale lengteschaal veel kleiner dan de relevante horizontale schalen. Hierdoor kan de druk in het water bij benadering direct worden berekend uit de waterstand. Aan de andere kant moet er in kustwateren terdege rekening worden gehouden met een groot aantal voor ingenieurs belangrijke processen, zoals bodemwrijving, droogvallen en onderlopen, dichtheidsverschillen en dammen en sluizen.

Er zijn verschillende typen modellen te onderscheiden. Voor een globale berekening van de afvoer van rivieren is het veelal voldoende een eendimensionaal model te gebruiken. Hierin wordt alleen gekeken naar de hoeveelheid water die per seconde door een bepaalde doorsnede stroomt, en niet naar de richting van de stroming.

Voor kustwateren en estuaria is minimaal een tweedimensionaal model nodig, waarin ook de (horizontale) richting van de stroming in het model meegenomen wordt. De waterhoogtevoorspelling aan

de Nederlandse kust kan goed met een dergelijk 2D-model gedaan worden. In driedimensionale modellen ten slotte kan ook verticale stroming worden berekend en kunnen situaties worden gemodelleerd waarbij het water op verschillende dieptes andere snelheden heeft. Zo is de stroomsnelheid nabij de bodem meestal een stuk kleiner dan aan het oppervlak, en door verticale gelaagdheid kan de stroming op verschillende dieptes zelfs tegengesteld gericht zijn. De effecten hiervan zijn bijvoorbeeld belangrijk bij het modelleren van zoutindringing in het Haringvliet.

Voor het doorrekenen (simuleren) van de wiskundige ondiepwatermodellen wordt het wateroppervlak overdekt met een rekenrooster, ofwel opgedeeld in vakjes. Per vakje wordt een soort gemiddelde waterstand en stroomsnelheid aangehouden. Hoe kleiner het vakje is, hoe nauwkeuriger deze gemiddelden overeenkomen met de werkelijke waarden.

In WAQUA/TRIWAQ is het mogelijk om een kromlijnig rooster te gebruiken, waardoor de afmetingen van de vakken niet op iedere plaats even groot hoeven te zijn. Op deze manier kan op plaatsen waar een complexer stroombeeld wordt verwacht, hogere resolutie worden verkregen. Ook kan hierdoor de kustlijn beter worden benaderd. In de simulatie worden al gauw zo'n 20.000 vakken gebruikt en worden voor ieder van de vakjes zo om de paar minuten lokale waterstanden en stroomsnelheden berekend.

Het simuleren van de modellen kan hierdoor op krachtige werkstations dagen rekenwerk vergen. De beschikbare rekenkracht vormt een belemmering voor toepassing van de simulatiemodellen.

2.2 Numerieke methoden in WAQUA/TRIWAQ

Aan de basis van de numerieke behandeling van ondiepwaterstroming in WAQUA en TRIWAQ staan de ondiepwatervergelijkingen. In het geval van WAQUA zijn dit de 2D dieptegemiddelde ondiepwatervergelijkingen, in TRIWAQ worden de 3D vergelijkingen gebruikt.

Ten eerste is er de continuïteitsvergelijking, die na discretisatie de massabalans per roostercel beschrijft:

De eerste term is de variatie van de waterstand ζ met de tijd, de tweede en derde termen zijn respectievelijk de verschillen tussen de massa-fluxen rechter-linkerzijde van de roostercel (x-richting) en boven-onder (y-richting).

Op een vergelijkbare manier worden er vergelijkingen voor de stroomsnelheden opgesteld, namelijk via een impulsbalans. De

hoeveelheid impuls in een controlevolume verandert in de tijd door meevoering (advectie), door het waterstandsverhang, de Corioliskracht, door turbulente uitsmering (viscositeit), en door wind en bodemwrijving. In twee dimensies (dieptegemiddeld) levert dit de volgende impulsvergelijkingen:

Dit zijn partiële differentiaalvergelijkingen met als primaire onbekenden de waterstand ζ als functie van plaats (x,y) en tijd (t), de horizontale snelheden U en V(x,y,t) (WAQUA, dieptegemiddeld) of u en v(x,y,z,t) (TRIWAQ), en de verticale snelheid w(x,y,z,t) (alleen in TRIWAQ).

De differentiaalvergelijkingen worden in WAQUA/TRIWAQ gediscretiseerd op een kromlijnig rooster met behulp van de eindige differentiemethode. Dit betekent dat de differentialen hierboven worden benaderd met differenties (verschillen) van de verschillende grootheden in verschillende naburige roosterpunten. Voor de tijdsintegratie wordt een zogeheten “Alternating Direction Implicit”-methode gebruikt (ADI). Effectief leiden deze technieken tot de volgende globale oplosprocedure:

voor tijd $t = T_{start}$ tot T_{eind} stap Δt doe

berekeningen eerste halve tijdstap:

- stelsel vergelijkingen voor $V(t + \Delta t/2)$
- stelsel vergelijkingen voor $U(t + \Delta t/2), \zeta(t + \Delta t/2)$

berekeningen tweede halve tijdstap:

- stelsel vergelijkingen voor $U(t + \Delta t)$
- stelsel vergelijkingen voor $V(t + \Delta t), \zeta(t + \Delta t)$

Een simulatie bestaat dus uit tijdstappen, een tijdstap bestaat uit verschillende fases die volgen uit de gebruikte ADI-methode, en per fase wordt er eerst een stelsel (algebraïsche) vergelijkingen opgesteld en dan wordt dit stelsel opgelost. De op te lossen stelsels hebben steeds een speciale structuur die volgt uit de gehanteerde ruimtelijke discretisaties.

Er worden speciale iteratieve oplosmethoden gebruikt die zijn toegespitst op de eigenschappen van de stelsels die worden opgelost. Een aantal van deze solvers zijn te vergelijken met

expliciete tijdsintegratiemethoden: stappen die onafhankelijk kunnen worden uitgevoerd voor alle roosterpunten. Andere solvers gebruiken het direct oplossen van tridiagonale stelsels en zijn daardoor minder gemakkelijk te parallelliseren. Hier wordt verder op ingegaan in module 3 van deze cursus. In alle gevallen is de hoeveelheid rekenwerk evenredig met het aantal roosterpunten dat wordt gebruikt in het gebiedsmodel. Dit vormt de basis voor de parallellisatie van WAQUA/TRIWAQ, waarbij wordt uitgegaan van een opdeling van het ruimtelijke rekenrooster over verschillende processoren.

3 Achtergronden van Parallel Rekenen

3.1 Parallelle Computers en clusters

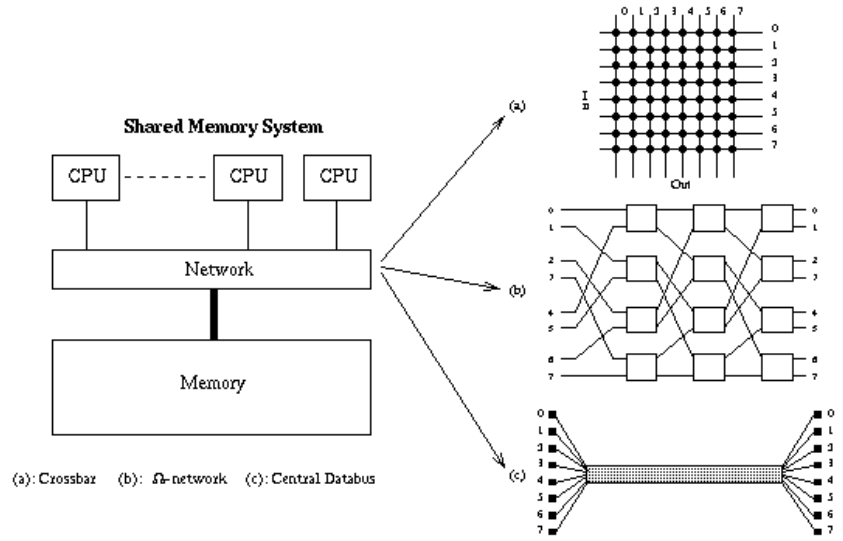
Onder een *parallelle computer* wordt over het algemeen een computer verstaan waarin zich meerdere CPU's bevinden, die door een enkel operating systeem beheerd worden. Daarnaast kent men (zeker tegenwoordig) ook een type parallelle rekenvoorziening dat bestaat uit een aantal min of meer volwaardige computers, elk met hun eigen operating systeem. Dit type computer wordt gewoonlijk aangeduid met de term *cluster*. In dit verband wordt soms gesproken over een *Beowolf cluster*, waarmee ruwweg een cluster wordt bedoeld dat geheel uit algemeen verkrijgbare hardware is opgebouwd.

Binnen de categorie van parallelle computers onderscheidt men verschillende typen op basis van de manier waarop de processoren en het werkgeheugen met elkaar verbonden zijn.

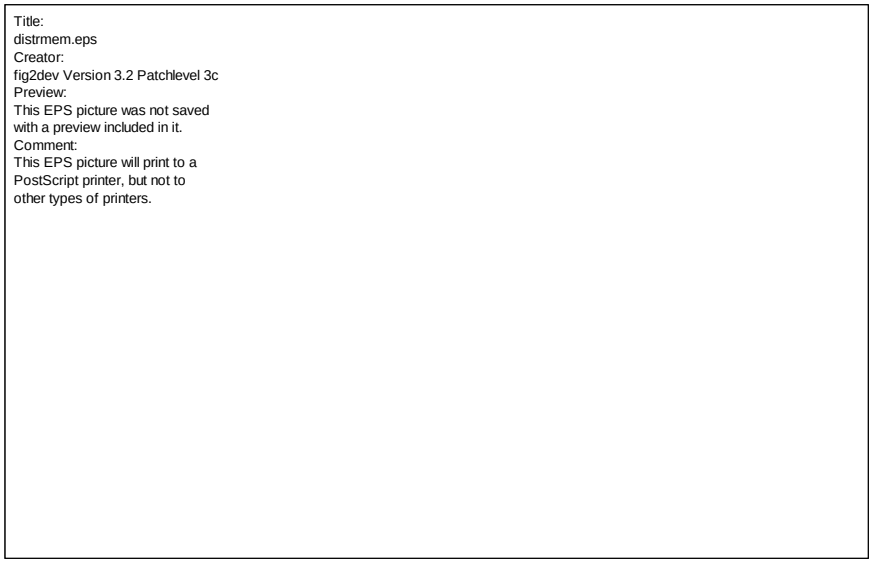
Enerzijds zijn er *shared memory* computers, waarbij alle processoren toegang hebben tot één enkel, centraal werkgeheugen. Daarnaast komt men ook gecombineerde modellen tegen waarbij de processoren zijn onderverdeeld in groepen die elk voor zich een shared memory hebben.

Anderzijds kent men *distributed memory* computers, dat wil zeggen: elke processor heeft een eigen werkgeheugen. Clusters behoren typisch tot de categorie van distributed memory computers. In een puur distributed memory systeem moeten processoren expliciet boodschappen met elkaar uitwisselen om informatie van elkaar te betrekken. Op moderne supercomputers hebben processoren meestal ook toegang tot de geheugens van andere processoren, zij het dat die toegang trager is dan de toegang tot hun eigen geheugen. Met cachegeheugens kan een deel van die vertraging ondervangen worden.

Bij distributed memory computers wordt ook een onderscheid gemaakt op basis van processor *topologie*. De processoren kunnen gerangschikt zijn in een (3D)-*mesh*, in een torus of in een *hypercube* topologie. Bij oudere modellen was het soms van belang om op de topologie te letten om een goede performance te kunnen halen. Bij moderne supercomputers speelt dit minder.



Figuur 4 : structuur van shared memory parallelle computers, en mogelijke structuren voor de interconnectie van processoren met het gezamenlijk geheugen



Figuur 5 : structuur van gedistribueerd geheugen parallelle computers, en mogelijke netwerktopologien voor de verbinding van de verschillende processoren

Een classificatie van parallelle computers die wat in onbruik is geraakt is de verdeling in *SIMD* (Single Instruction, Multiple Data) en *MIMD* (Multiple Instruction, Multiple Data) machines. De

eerste categorie omvatte machines waarin er slechts één computerinstructie tegelijk werd uitgevoerd, maar dan wel voor een groot aantal data elementen tegelijk. De tweede categorie omvatte de machines waarin echt meerdere volwaardige processoren aanwezig waren. Omdat op dit moment eigenlijk uitsluitend MIMD computers worden gemaakt, heeft deze classificatie weinig zin meer.

Een belangrijke eigenschap van parallelle computers is de *schaalbaarheid*. Een parallelle computer is goed schaalbaar als hij kan worden uitgebreid met meer processoren zonder dat de prestaties per processor afnemen. Echte shared memory computers hebben doorgaans meer problemen met schaalbaarheid omdat het bij zeer grote aantallen processoren lastig wordt om ze allemaal gelijktijdig toegang tot het werkgeheugen te geven.

Vaak werken gebruikers niet rechtstreeks op een parallelle computer. In plaats daarvan bereiden ze een berekening voor op een *front-end*, d.i. een relatief normale computer. Als de berekening dan gedaan moet worden zal het operating systeem van het front-end ervoor zorgen dat die wordt uitgevoerd op de parallelle computer.

3.2 Performance maten

Parallelle computers worden uiteindelijk alleen ingezet om een versnelling te bereiken ten opzichte van het gebruik van een conventionele computer, of om grotere problemen te kunnen oplossen dan anderszins mogelijk is (bijv. qua geheugenomvang).

Deze versnelling kan op een aantal manieren worden weergegeven. De meest gebruikte maat is de zogeheten *speedup*. Als men gebruik maakt van p processoren, dan wordt de speedup berekend door de rekentijd op 1 processor te delen door die op p processoren:

$$S(p) = \frac{T_1}{T_p}$$

Merk op dat de speedup een functie is van p. Als de rekentijd per processor precies een factor p kleiner is van T_1 dan volgt hiermee $S(p)=p$. In dit geval spreekt men van *lineaire speedup*, hoewel die term wiskundig gezien ook gebruikt zou kunnen worden voor andere situaties waarin $S(p)=a.p$ met $a \neq 1$. Er wordt gesproken van *superlineaire speedup* als $S(p)>p$.

De *efficiëntie* is een maat voor de gemiddelde versnelling per processor en is gedefinieerd als:

$$E(p) = \frac{S(p)}{p}$$

De efficiëntie ligt normaalgesproken tussen 0 en 1, maar kan groter dan 1 zijn in het geval van superlineaire speedup.

Doorgaans vlakt *S(p)* af naarmate *p* hoger wordt en uiteindelijk zal de functie ook weer gaan dalen: het gebruik van nog meer processoren werkt dan contraproductief.

Er zijn een aantal redenen waarom het niet mogelijk is om een onbeperkt aantal processoren in te zetten. In de eerste plaats is het vaak zo dat niet alle berekeningen parallel uitgevoerd kunnen worden. Stel dat een fractie *f* van de berekeningen geparallelliseerd kan worden, dan kan men in het ideale geval T_p schrijven als

$$T_p = \frac{fT_1}{p} + (1-f)T_1$$

waarmee de speedup als functie van *p* gegeven wordt door:

$$S(p) = \frac{T_1}{T_p} = \frac{p}{f + (1-f)p}$$

wat voor kleine aantallen processoren *p* nog vrijwel lineaire speedup geeft, maar voor grote aantallen processoren altijd zal naderen tot een limiet van *1/(1-f)*. Als *f* dicht in de buurt van 1 zit, kan dit nog altijd een aanzienlijke speedup zijn, maar als men eenmaal in de buurt van de limiet zit, dan heeft het geen zin om nog meer processoren in te zetten. Deze wetmatigheid staat bekend als de *wet van Amdahl*.

Ten tweede heeft men vaak te maken met interprocessor communicatie, vooral in het geval van echte distributed memory systemen. In dergelijke systemen zenden de processoren berichten naar elkaar. De parallelle rekentijd wordt dan:

$$T_p = \frac{fT_1}{p} + (1-f)T_1 + T_{comm}(p)$$

De tijd die nodig is om een bericht te sturen is gewoonlijk opgebouwd uit een deel dat onafhankelijk is van de lengte van het bericht, de *latency*, en een deel dat daar wel van afhankelijk is. De latency houdt verband met allerlei zaken rondom het opstarten van een boodschap, inclusief bijvoorbeeld het versturen van een boodschapsheader waarin staat hoeveel informatie er verder nog in de boodschap zit. De opstarttijd is gewoonlijk in de orde van 10-100 microseconden.

De snelheid waarmee informatie wordt verstuurd als de boodschap eenmaal op gang is, wordt meestal aangeduid met de term *bandbreedte*, gegeven in Mbyte/sec.

Ten slotte bevat de tijd voor het uitwisselen van een boodschap vaak wat *synchronisatie tijd*, dat is: tijd die nodig is omdat de versturende processor nog niet toe is aan het sturen van een boodschap: de ontvangende processor moet dan even wachten.

Voor het versturen van een boodschap van L bytes is dus een tijd T_{send} nodig van

$$T_{comm} = T_{startup} + L / B + T_{sync}$$

waarin B de bandbreedte is.

Een klein rekenvoorbeeld kan duidelijk maken dat de latency een niet te verwaarlozen aspect van de communicatie is. Stel dat de latency 100 microseconden is en de bandbreedte 100 Mbyte/sec, dan bestaat de tijd voor het versturen van 100 bytes voor 99% uit latency. Het is dus zaak om zoveel mogelijk informatie in één boodschap te versturen.

Indien het berichtenverkeer heel intensief wordt, dan kan het voorkomen dat het netwerk dit niet meer aan kan. De communicatie tijd wordt dan langer dan men zou verwachten. Men spreekt dan van *netwerkcontentie*, ofwel 'strijd' om het beschikbare netwerk.

Een laatste factor die vaak in beschouwing moet worden genomen bij het beoordelen van de parallelle performance is de *load balance*, ofwel de mate waarin alle processoren een gelijke hoeveelheid werk te doen krijgen. Stel bijvoorbeeld dat men een stuk rekenwerk verdeeld over twee processoren, waarvan de eerste 60% van het werk moet doen en de tweede 40%. Dan zal de speedup slechts 1.6 zijn in plaats van de maximaal haalbare 2. De tweede processor wordt eenvoudigweg onvolledig benut.

3.3 Programmeermodellen

Het is aan de programmeur om het rekenwerk dat gedaan moet worden te verdelen over de beschikbare processoren. Daarbij kan hij op verschillende manieren zijn programma ontwikkelen.

De eenvoudigste oplossing wordt geboden door programmeertalen die parallel rekenen ondersteunen. Zo kent men in Fortran90 bijvoorbeeld het statement de zogenaamde *array syntax*, waardoor men code kan schrijven als:

```
REAL,DIMENSION(10)::x,y
x=x+y
```

en de compiler vrij is om de berekening te verdelen over de beschikbare processoren. Deze manier van parallelliseren wordt aangeduid met de term *data-parallel*: er is één enkele instructie die tegelijkertijd op een groot aantal data-elementen moet worden toegepast.

Parallelliserende compilers maken voornamelijk gebruik van deze wijze van parallelliseren. Het voordeel van deze manier van werken is dat het programmeren op deze manier niet veel moeilijker is dan het schrijven van een gewoon (niet-parallel) programma. Bovendien kan de compilerbouwer optimaal gebruik maken van zijn kennis van de computerhardware om de berekeningen zo optimaal mogelijk aan processoren toe te wijzen. Anderzijds is het vaak erg lastig om op deze manier een hoge en schaalbare speedup te krijgen. De fractie niet- of niet-volledig geparallelliseerde berekeningen blijft vaak hoog.

Recentelijk is een veel krachtiger mechanisme beschikbaar gekomen om binnen één programma parallelle rekentechnieken te gebruiken. Dit mechanisme is *OpenMP* en biedt allerlei mechanismen om grote stukken rekenwerk als geheel parallel te laten uitvoeren. Dit is feitelijk geen data-parallelle manier van programmeren meer, maar biedt nog wel het voordeel dat de programmeur slechts binnen de context van een enkel programma hoeft te denken.

Naast de data-parallelle programmeerconstructies zoals “forall” heeft de programmeur in OpenMP ook de beschikking over allerlei constructies waarmee hij parallelle “threads” direct kan aansturen: “thread-private” variabelen, updates van “shared” variabelen die zijn beveiligd tegen gelijktijdige toegang door meerdere threads (atomische update), expliciete synchronisatieopties e.d. Het nadeel van deze constructies ten opzichte van de data-parallelle manier van werken is echter wel dat de programmeur veel meer werk moet doen. Hij moet nu zelf aangeven hoe het rekenwerk moet worden verdeeld over verschillende threads en moet goed op de synchronisatie van de verschillende berekeningen letten. Dit extra werk blijkt echter in de praktijk nodig om een goede performance te behalen, in het bijzonder voor het rekenen op grotere aantallen processoren.

De meest gangbare wijze van parallel programmeren is waarschijnlijk de manier die bekend staat als *SPMD*: Single Program, Multiple Data. De programmeur schrijft een enkel programma dat meerdere keren tegelijk wordt opgestart, elk met een eigen set aan gegevens om door te rekenen. De manier waarop WAQUA/TRIWAQ geparallelliseerd is, is in grote lijnen een SPMD benadering.

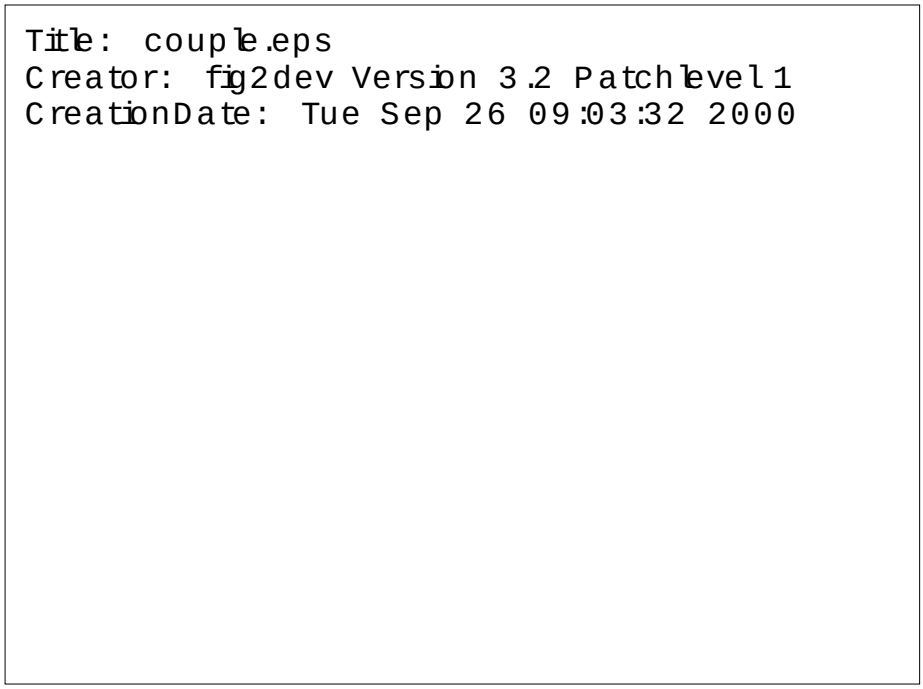
Gewoonlijk moeten de processen op de een of andere manier hun werk afstemmen, wat ze dan doen door het uitwisselen van gegevens. Voor deze gegevensuitwisseling maakt men gebruik van hoog-niveau communicatie bibliotheken. Tot voor kort was PVM (Parallel Virtual Machine) de meest gangbare van die bibliotheken, maar diens rol wordt in hoog tempo overgenomen door MPI (Message Passing Interface). Qua functionaliteit verschillen beide bibliotheken niet wezenlijk van elkaar.

Bij SPMD programmeren voeren alle verschillende processoren steeds hetzelfde programma uit. Uiteraard kan men ook meerdere aparte programma's schrijven, die elk een bepaalde berekening uitvoeren. Deze wijze van parallelliseren wordt vaak aangeduid als *functioneel parallel*: verschillende functies worden parallel aan elkaar uitgevoerd. Hoewel dit in zijn algemeenheid een lastige manier van programmeren is, komt men enkele speciale gevallen ervan toch vaak tegen.

De meest bekende variant is het *Master-Slave* of ook *Master-Worker* model: er is een masterproces dat het uitvoeren van tussenberekeningen uitbesteed aan slave- of werkerprocessen, die het resultaat terugsturen naar de master. De slaveprocessen zijn verder volledig passief. Strikt genomen behoort parallel WAQUA/TRIWAQ tot deze categorie van parallelle programma's. Maar omdat de master (COEXEC) voor WAQUA/TRIWAQ slechts een zeer beperkte functie heeft, is het evengoed gerechtvaardigd om parallel WAQUA/TRIWAQ een SPMD programma te noemen.

Een andere bekende variant van functioneel parallellisme is de *Pipeline*. De processen staan als het ware in serie geschakeld en elk proces ontvangt input van de ene buurman en stuurt het resultaat van zijn berekening naar de andere buurman. Op deze vorm van parallellisme waren de *vectorcomputers* gebaseerd, die tot enkele jaren geleden de markt voor supercomputers domineerden: de verschillende functies waren daarbij in de hardware ingebouwd.

3.4 Globaal ontwerp: COPPRE/COPPOS, COCLIB



Figuur 6: Globale opzet van parallel WAQUA/TRIWAQ

Figuur 6 toont schematisch de structuur van WAQUA/TRIWAQ. De delen die met oranje zijn aangegeven zijn de stukken die de parallelle functionaliteit verzorgen. De niet-oranje stukken komt men ook bij gewone, niet-parallelle WAQUA/TRIWAQ simulaties tegen.

Voor de gebruiker is het werken met parallel WAQUA/TRIWAQ niet veel anders dan wanneer men geen gebruik maakt van parallel rekenen. Men begint met het maken van een invoer-(siminp-)file en maakt daar met behulp van WAQPRE een SDS-file van. Vervolgens roept men het runscript `WAQPRO.RUN` aan, waarbij men in het geval van een parallelle run een aantal processoren groter dan 1 invult.

3.4.1 Partitioner COPPRE

Het runscript neemt een groot deel van de complexiteit van parallel rekenen voor zijn rekening. Allereerst wordt de SDS-file opgeknipt in een aantal kleine SDS-files, één voor elk proces. Deze opsplitsing gebeurt door het programma COPPRE (ook wel de *Partitioner* genoemd). Elk van de resulterende SDS-files is in feite een volledige valide SDS-file voor een deel van het rekenrooster. COPPRE is een snel programma. Zelfs voor grote SDS-files heeft het doorgaans niet veel meer dan een minuut nodig. Wel kan het

gebeuren dat COPPRE flink wat geheugen nodig heeft om heel grote arrays in core te kunnen hebben. Het runscript biedt dan ook de mogelijkheid om de door COPPRE te gebruiken buffergrootte expliciet op te geven.

De opdeling van het rekenrooster dat ten grondslag ligt aan de opsplitsing van de SDS-file, wordt normaalgesproken door COPPRE zelf bepaald. Daarbij zal COPPRE proberen om elke processor evenveel werk te doen te geven (d.i. een goede load balance creëren) en tegelijk proberen om de hoeveelheid gegevens die tussen de processen moet worden uitgewisseld te minimaliseren. COPPRE kent twee manieren om een dergelijke verdeling te maken. De eerste manier is de ORB (Orthogonal Recursive Bisection) methode. Deze bestaat uit het om beurten opdelen van het rekenrooster langs de x-richting en y-richting en is geschikt voor veel WAQUA/TRIWAQ gebiedsschematisaties. De tweede methode gebruikt alleen opdeling langs de x-richting of de y-richting (stripsgewijs), en is geschikt voor langgerekte gebiedsschematisaties zoals het Maasmodel.

Als de performance kritisch is, kan het zijn dat men als gebruiker geen genoegen neemt met de automatisch bepaalde opdeling van het rekenrooster. In dat geval heeft men de mogelijkheid om de opdeling expliciet aan COPPRE op te geven door middel van een partitie invoerfile. Als COPPRE een partitie invoerfile binnenkrijgt, zal hij die gebruiken en niet zelf nog eens een andere opdeling bepalen.

Partitie invoerfiles kunnen met de hand gemaakt worden, maar er is ook een handige tool voor beschikbaar, genaamd VISIPART. Dit is een Matlab programma waarmee men grafisch een opdeling van het rekenrooster kan. Daarnaast kan VISIPART gelijk een schatting geven de de load balance.

COPPRE bepaalt niet alleen de opsplitsing van het rekenrooster, maar splitst ook alle arrays op de SDS-file op op basis van de verdeling van het rekenrooster. Daarvoor heeft hij enige kennis nodig van de array's op de SDS-file en hun structuur. Deze kennis wordt toegeleverd middels een Data Description File. Deze file zal in principe nooit door een gebruiker gezien worden, maar kan wel worden aangepast als de inhoud van SDS-files verandert (bijvoorbeeld doordat er nieuwe functionaliteit in WAQPRO wordt ingebouwd). Mits de verandering van de inhoud van SDS-files niet al te ingrijpend is, kan men op die manier COPPRE daarop aanpassen zonder dat er echt geprogrammeerd hoeft te worden.

Zodra COPPRE zijn werk gedaan heeft, zijn er in de werkdirectory dus evenveel kleine SDS-files ontstaan als er processen gestart gaan worden. Deze kleine SDS-files hebben de naam van de

originele SDS-file, met daarachter een 3-cijferig nummer, beginnend bij 001. Daarnaast is er nog een SDS-file met het nummer ‘000’ achter de naam. Dit is een file die niet tijdens het rekenen gebruikt wordt, maar dient om allerlei tabellen door te geven van COPPRE naar COPPOS, waarover later meer.

3.4.2 Masterprogramma COEXEC

Nadat COPPRE klaar is, wordt het masterprogramma COEXEC opgestart. Dit programma doet weinig meer dan het opstarten van de WAQUA/TRIWAQ processen. Nadat de processen eenmaal zijn opgestart, bestaat de taak van COEXEC uitsluitend in het afhandelen van berichten van de rekenprocessen. De standaard-uitvoer van die processen wordt ook in de vorm van een bericht naar COEXEC gestuurd; COEXEC fungeert dus ook als het luik waardoorheen de rekenprocessen printuitvoer genereren.

Naast printuitvoer kan COEXEC ook meer inhoudelijke boodschappen van de processen ontvangen, zoals een mededeling dat een communicatie tussen twee rekenprocessen niet vlot genoeg verloopt. COEXEC verzamelt dit soort storingsberichten en kan uiteindelijk besluiten dat de run niet goed verloopt en afgebroken moet worden. Dit gebeurt in praktijk niet vaak en altijd om zeer goede redenen.

3.4.3 Communicatiebibliotheek COCLIB

De rekenprocessen zijn elk een normaal WAQUA/TRIWAQ proces (WAQPRO.EXE). Wel is het zo dat de parallelle WAQUA/TRIWAQ processen elk een kleiner probleem door te rekenen hebben, zodat ze gewoonlijk met een kleinere buffergrootte toe kunnen.

De rekenprocessen communiceren met elkaar via een hoog-niveau communicatie bibliotheek, COCLIB genaamd. Deze bibliotheek is gebouwd op basis van PVM en verbergt alle onnodige details van de communicatie voor de programmeur. De programmeur hoeft uitsluitend te specificeren wat hij op een bepaald punt moet ontvangen en wat hij kan verzenden; COCLIB zoekt uit welke informatie naar welke processen gezonden moet worden.

Zoals gezegd, is COCLIB gebaseerd op PVM. Deze keuze is niet heel essentieel; PVM zou relatief gemakkelijk te vervangen moeten zijn door MPI. Tot op heden wordt PVM echter nog goed ondersteund, dus er is geen onmiddellijke noodzaak om over te stappen.

Een van de hebbelijkheden van PVM is dat het een enkele keer voorkomt dat een run die niet op een normale manier eindigt wat vervuiling achterlaat, waardoor nieuwe runs niet kunnen opstarten.

Voor dergelijke situaties kent het runscript `WAQPRO.RUN` de optie 'Force_pvm': als deze optie aanstaat, zal de vervuiling eerst worden opgeruimd.

3.4.4 Collector COPPOS

Elk rekenproces werkt direct op zijn eigen SDS-file. Ook de uitvoer van het rekenproces (zoals MAPS en TIME-HISTORIES) worden direct naar de eigen SDS-file geschreven. Dat betekent dat na afloop van een run de resultaten verspreid staan over de kleine SDS-files. Het programma COPPOS (de tegenhanger van COPPRE en ook wel aangeduid als de *Collector*) raapt deze resultaten bijeen en voegt ze toe aan de originele SDS-file. Na afloop van de run worden de kleine SDS-files normaalgesproken opgeruimd en staan alle resultaten in de origineel toegeleverde SDS-file.

3.5 Performance Aspecten

De performance van parallel WAQUA/TRIWAQ wordt in grote lijnen bepaald door de eigenschappen van de machine waarop gewerkt wordt. Daarnaast kan men door het maken van een geschikte partitionering de performance verbeteren. In deze paragraaf zal daar nader op in gegaan worden.

Een geschikte partitionering zorgt ervoor dat elke processor evenveel werk te doen heeft en dat er niet teveel informatie hoeft te worden uitgewisseld tussen de processen onderling.

Een eenvoudig model voor een parallelle berekening dat goed voldoet voor het analyseren van de performance is dat alle processoren tegelijk aan het communiceren zijn (BSP, bulk synchronous parallel). Onder deze aanname kan de rekentijd van een parallelle som worden uitgesplitst in de tijd voor de rekenfases versus de tijd voor de communicatiefases.

$$T_p = T_{calc}(p) + T_{comm}(p)$$

De tijd voor de rekenfases wordt bepaald door de processor die het meeste werk heeft te doen. Het is hierbij zinvol om deze tijd te relateren aan de gemiddelde rekentijd per processor, en om de som over alle processoren te relateren aan de rekentijd die een enkele processor nodig heeft voor de totale berekening:

$$T_{calc}(p) = \max_{p'} T(p')$$
$$= \frac{p \cdot \max_{p'} T(p')}{\sum_{p'} T(p')} \cdot \frac{\sum_{p'} T(p')}{T_1} \cdot \frac{T_1}{p}$$
$$= \alpha \cdot \beta \cdot \frac{T_1}{p}$$

De parameters α en β hierin zijn respectievelijk een maat voor de synchronisatie overhead en voor de numerieke overhead van de berekening op *p* processoren.

Een voorbeeld van een bron van numerieke overhead is dat er soms meer iteraties nodig zijn voor een bepaalde procedure naarmate er meer processoren worden gebruikt. Aan de andere kant kunnen berekeningen ook sneller gaan verlopen naarmate er meer processoren worden ingezet ($\beta<1$) bijvoorbeeld ten gevolge van cache-effecten: het geheugen van moderne computers is onderverdeeld in stukken met verschillende snelheden (registers, cache, main memory, swap/disk), en de grootte en vorm van deelroosters bepaalt mede de mate waarin het cache geheugen efficiënt kan worden gebruikt.

Door toedoen van numerieke overhead is het ervoor zorgen dat elke processor evenveel werk te doen heeft niet slechts een kwestie van elke processor evenveel roosterpunten toewijzen. Het is daarom vaak nodig om enkele korte runs te doen om te bepalen hoe de load-balance daadwerkelijk uitpakt. Uit bovenstaande formule valt af te leiden dat het doel hierbij is om een *zo klein mogelijke grootste rekentijd* te bereiken. In plaats van een deeldomein dat veel kleiner is dan gemiddeld te vergroten kan men dus beter de aandacht richten op het verkleinen van de grootste deeldomeinen.

De communicatietijd $T_{comm}(p)$ bestaat uit een stuk latency en een stuk dat direct afhankelijk is van de hoeveelheid gegevens die uitgewisseld worden (zie paragraaf 3.2). Als de hoeveelheid communicatie per processor sterk verschilt dan komt daar bovendien nog een stuk synchronisatietijd bij, het is (in het BSP-model) wederom het maximum van de communicatietijd over alle processoren dat maatgevend is.

Het latency stuk kan men minimaliseren door te zorgen dat er zo min mogelijk boodschappen verstuurd hoeven te worden, dus: er voor te zorgen dat elke stuk van het rekenrooster aan zo min mogelijk andere stukken grenst.

De hoeveelheid uit te wisselen gegevens kan men minimaliseren door ervoor te zorgen dat de grenzen ofwel *interfaces* tussen de verschillende stukken van het rekenrooster zo klein mogelijk zijn.

De hoeveelheid uitgewisselde gegevens is namelijk bij benadering evenredig aan het aantal (natte) roosterpunten op de interfaces tussen verschillende stukken van het rekenrooster.

Ten slotte zijn er situaties waarin *netwerkcontentie* van belang is: de communicaties van verschillende processoren moeten op elkaar wachten omdat ze van een gezamenlijke resource gebruik moeten maken. Dit treedt bijvoorbeeld op bij een traditioneel Ethernet met een busstructuur, of bij gebruik van een stervorm op basis van een hub in plaats van een switch. Wanneer er in een dergelijk geval meerdere computers gelijktijdig beginnen met communiceren, dan kan het netwerk die plotselinge belasting niet goed aan. In de gepresenteerde formules kan dit worden verwerkt door de tijd per byte te verkleinen: 1/effectieve bandbreedte per proces. In deze situaties kan het soms voordelig zijn om de load-balance met opzet iets te verstoren. Hierdoor wordt de communicatie enigszins gespreid in de tijd, en wordt er efficiënter gebruik maakt van het netwerk.

Bij het optimaliseren van roosteropdelingen is de load balance in het algemeen belangrijker dan de hoeveelheid communicatie. Dit geldt sterker naarmate er met grotere modellen en minder processoren wordt gewerkt. Dit komt doordat de synchronisatietijd min of meer proportioneel is met de oppervlakte van deeldomeinen, terwijl de communicatietijd is gerelateerd aan de lengte van de randen van deeldomeinen.

3.6 Nauwkeurigheidsaspecten

De nauwkeurigheid van simulaties met WAQUA/TRIWAQ met gebruik van parallel rekenen is hetzelfde als die van simulaties op een enkele processor. Dat is omdat er bij de parallellisatie geen concessies zijn gedaan met betrekking tot de gebruikte numerieke methoden. In principe zou de gebruiker niets van het parallel rekenen moeten merken, behalve dat de berekeningen een stuk sneller worden uitgevoerd.

In de praktijk blijken er af en toe toch forse verschillen te ontstaan (lokaal/tijdelijk centimeters verschil) tussen een sequentiële en een parallelle run met dezelfde invoer. Dit ligt hoofdzakelijk aan de gevoeligheid van het model en/of de schematisatie. In een parallelle run worden wel dezelfde berekeningen uitgevoerd, maar niet per sé in precies dezelfde volgorde. Hierdoor kunnen afrondprocessen ten gevolge van het gebruik van enkele precisie toch kleine verschillen opleveren tussen de verschillende runs. Zulke minieme verschillen kunnen leiden tot net wel of net niet droogvallen van een punt, waardoor de verschillen sterk kunnen worden vergroot. Een ander aspect dat een rol speelt bij de

gevoeligheid van schematisaties is het optreden van verticale gelaagdheid.

Naast afrondfouten worden kleine verschillen ook geïntroduceerd door de gebruikte iteratieve procedures voor het oplossen van stelsels vergelijkingen. Vooral bij het oplossen van de tridiagonale stelsels voor de continuïteitsvergelijking wordt er per iteratie in een parallelle berekening iets anders gedaan dan in een sequentiële som. Ook het gebruikte aantal processoren en de roosteropdeling zijn hier van belang. Maar het eindresultaat na convergentie van het proces is wel hetzelfde. De grootte van de verschillen kan worden verkleind door het instellen van een groot aantal iteraties.

Een heel ander soort verschillen betreft de verschillen tussen sequentiële runs met de niet-parallelle versies van WAQUA (tot half 1999) en de parallelle versies van WAQUA (vanaf medio 1999). Om parallellisatie van WAQUA mogelijk te maken is het numerieke model van WAQUA enigszins aangepast. Twee relevante wijzigingen in dit kader zijn een aanpassing van het droogvalalgoritme, en het verwijderen van de viscositeits-kruistermen uit de implementatie. Beide aanpassingen zijn uitgebreid gevalideerd. De eerste leidt lokaal en tijdelijk tot centimeters verschil in modellen waarin droogvallen een belangrijke rol speelt. De tweede aanpassing heeft in sommige riviersituaties een grote invloed op het verhang dat uiteindelijk wordt voorspeld (verschil tot 1cm/km), en dit blijkt een onvolkomenheid te zijn in de oorspronkelijke implementatie.

3.7 Herstructurering van sequentieel WAQUA

De sequentiële versie van WAQUA is uitgebreid geherstructureerd ten behoeve van de parallellisatie. Daarbij zijn een paar wijzigingen in het numerieke model doorgevoerd (zie ook hierboven), maar vooral is de implementatie sterk aangepast. De aanpassingen zijn uitgebreid gevalideerd, waarbij ook de performance-effecten zijn meegenomen (zie VORtech Computing rapporten TR98-02, 98-05 en 99-04).

De noodzaak voor de herstructurering volgt uit de eisen die parallel rekenen stelt aan de gebruikte rekenmethode:

1) Er moeten voldoende berekeningen zijn die tegelijkertijd mogen worden uitgevoerd (parallellisme).
2) De berekeningen moeten zodanig kunnen worden verdeeld over meerdere processoren dat er niet te vaak en te veel gegevens moeten worden uitgewisseld (granulariteit, overhead).

Berekeningen mogen alleen tegelijkertijd worden uitgevoerd als het resultaat daardoor niet verandert, als ze niet afhankelijk van elkaar zijn. Dit speelde een rol in de oplosmethode voor de continuïteitsvergelijking voor WAQUA. Deze methode werkte vroeger voor iedere rij van het rekenrooster afzonderlijk:

voor rij = 1, .., norows doe
- voer droogval/onderloop-controles uit
- bepaal coefficienten impuls/continuiteitsvgl.
- los stelsel op met iteratieve procedure en met droogval-controles

In dit schema zijn de berekeningen van verschillende rijen van elkaar afhankelijk door de droogvalcontroles. Namelijk, de coëfficiënten van de impulsvergelijking hangen af van de actuele geometrie in naastgelegen rijen. Dus als er in rij 1 iets droogvalt dan heeft dat effect op rij 2, en dus kunnen de berekeningen voor rijen 1 en 2 niet tegelijkertijd worden uitgevoerd. Vanwege deze afhankelijkheid had de oorspronkelijke methode te weinig parallellisme: alle processoren zouden steeds samen aan een enkele rij moeten werken, en per rij is er niet zo heel veel gelijktijdig te doen.

Wanneer de processoren toch samen aan een rij gaan werken, dan treedt meteen het tweede probleem op. Ze moeten dan ontzettend vaak met elkaar gaan communiceren, voor iedere iteratie voor iedere rij afzonderlijk. Vanwege de opstarttijd van communicaties in parallelle computers (de latency, zie paragraaf 3.2) zou er vrijwel geen enkele versnelling mogelijk zijn.

Om dit te verbeteren wordt momenteel het volgende schema

voor rij = 1, .., norows doe
- voer droogval/onderloop-controles uit

voor rij = 1, .., norows doe
- bepaal coefficienten impuls/continuiteitsvgl.

voor rij = 1, .., norows doe
- los stelsel op met iteratieve procedure en met droogval-controles

gebruikt:

Dit schema leidt tot iets andere rekenresultaten doordat er bij het bepalen van coëfficiënten met een iets andere geometrie wordt

gewerkt. Maar de wijziging is niet fundamenteel: beide geometriën kunnen in principe even goed worden gebruikt.

Het oorspronkelijke schema is in WAQUA geïntroduceerd omdat het weinig werkgeheugen vereist en ook tot een goede performance leidt. Bovengenoemde herstructurering ten behoeve van parallel rekenen zorgt dus voor een toename van de rekentijd en het geheugenbeslag van niet-parallelle sommen. Om dit tegen te gaan (vooral het performanceverlies) zijn extra wijzigingen ingevoerd die de performance juist verbeteren:

- Er is een optie ingevoerd om het stopcriterium voor de continuïteitsvergelijking te baseren op waterstanden in plaats van stroomsnelheden. De stroomsnelheden moeten in iedere iteratie alleen worden uitgerekend ten behoeve van het stopcriterium, dit wordt uitgespaard met de nieuwe optie.
- De viscositeits-kruistermen zijn optioneel gemaakt. Dit zijn termen die wel flink wat rekentijd kosten, maar die meestal weinig effect hebben op de waterbeweging. In bepaalde situaties waarin ze wel een groot effect hebben kan worden aangetoond dat dit effect ongewenst is.
- Een groot aantal kleine verbeteringen, zoals het opslaan van rooster-transformatiecoëfficiënten in semi-permanente arrays in plaats van ze steeds opnieuw te berekenen, en zoals het in aparte loops uitrekenen van wind- en dichtheidseffecten.

Experimenten op een HP werkstation hebben uitgewezen dat WAQUA door de herstructurering eerder sneller dan langzamer is geworden. Van de zeven gebruikte testmodellen gaven er zes een verbetering te zien, en slechts één een toename van de rekentijd.

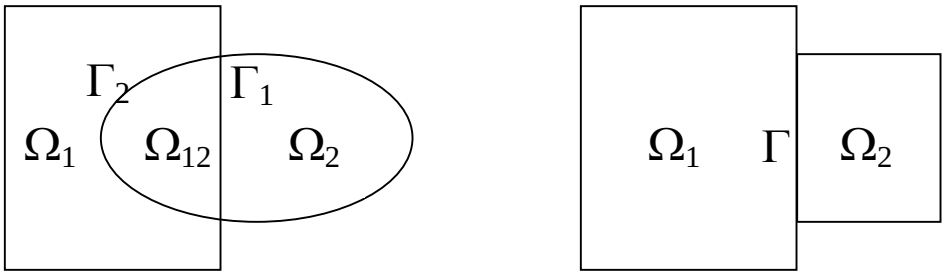
4 Achtergronden van Domein Decompositie

4.1 Algemene introductie Domein Decompositie

Domein decompositie is een term waarmee een scala aan technieken wordt aangeduid waarbij de oplossing van een partiële differentiaal vergelijking wordt gevonden door het domein van de vergelijking op te delen en vervolgens per deeldomein het probleem op te lossen. Het oplossen per deeldomein gebeurt op een zodanige manier dat de combinatie van de oplossingen voor de deeldomeinen de oplossing voor het gehele domein vormt. De rekenroosters van de deeldomeinen kunnen in principe onafhankelijk van elkaar worden gemaakt. Het rooster van het ene deeldomein kan dus fijner zijn dan dat van een andere of het kan een andere oriëntatie hebben. Ook hoeven de roosters in principe niet op elkaar aan te sluiten: ze mogen ook gedeeltelijk overlappen.

De uitvinding van domein decompositie wordt gewoonlijk toegeschreven aan H.A. Schwarz, die in 1870 een dergelijke techniek gebruikte om aan te tonen dat er harmonische functies bestaan op onregelmatige domeinen die een samenstelling zijn van overlappende deeldomeinen.

Binnen de klasse van domein decompositie methoden wordt vaak onderscheid gemaakt tussen overlappende en niet-overlappende domein decompositie.



In bovenstaande figuur is links de situatie weergegeven waarbij deeldomeinen Ω_1 en Ω_2 overlappen en beiden gebied Ω_{12} omvatten. Een deel van de rand $\partial\Omega_1$ van deeldomein Ω_1 ligt in deeldomein Ω_2; dit deel wordt aangeduid met Γ_1. Hetzelfde geldt m.m. voor de rand van deeldomein Ω_2. In de rechterfiguur is een niet-overlappende domein decompositie weergegeven. De beide deeldomeinen overlappen niet, maar delen een stuk van hun rand. Dit gedeelde stuk rand wordt weergegeven met Γ.

In het algemeen zoekt men nu op het samengestelde domein Ω de oplossing van een differentiaal vergelijking:

$$Lv = f \quad \text{op} \quad \Omega$$

waarin L de differentiaal operator is en v de gezochte onbekende.

Deze oplossing kan, in het geval van overlappende domein decompositie bijvoorbeeld verkregen worden door het iteratief oplossen van de volgende twee subproblemen:

$$\begin{cases} Lv_1^{k+1} = f & \text{op} \quad \Omega_1 \\ v_1^{k+1} = v^k & \text{op} \quad \Gamma_1 \\ v_1^{k+1} = g & \text{op} \quad \partial\Omega_1 \setminus \Gamma_1 \end{cases}$$

en

$$\begin{cases} Lv_2^{k+1} = f & \text{op} \quad \Omega_2 \\ v_2^{k+1} = v^k & \text{op} \quad \Gamma_2 \\ v_2^{k+1} = g & \text{op} \quad \partial\Omega_2 \setminus \Gamma_2 \end{cases}$$

waarbij de oplossing op het totale domein dan gegeven wordt door

$$v^{k+1}(x,y) = \begin{cases} v_1^{k+1}(x,y) & \text{als } (x,y) \in \Omega \setminus \Omega_2 \\ v_2^{k+1}(x,y) & \text{als } (x,y) \in \Omega_2 \end{cases}$$

Onder bepaalde voorwaarden zal deze iteratieve oplossing uiteindelijk convergeren naar de gezochte oplossing.

Bij niet-overlappende domein decompositie kan men iets soortgelijks doen, waarbij dan de twee deelproblemen bijvoorbeeld gegeven kunnen zijn door:

$$\begin{cases} Lv_1^{k+1} = f & \text{op} \quad \Omega_1 \\ a_{links} \dfrac{\partial v_1^{k+1}}{\partial n} = a_{rechts} \dfrac{\partial v_2^{k}}{\partial n} & \text{op} \quad \Gamma \\ v_1^{k+1} = g & \text{op} \quad \partial\Omega_1 \setminus \Gamma \end{cases}$$

en

$$\begin{cases} Lv_2^{k+1} = f & \text{op} \quad \Omega_2 \\ v_2^{k+1} = v_1^{k} & \text{op} \quad \Gamma \\ v_2^{k+1} = g & \text{op} \quad \partial\Omega_2 \setminus \Gamma \end{cases}$$

waarin n de naar buiten gerichte normaal van de rand van deeldomein Ω_1 is.

Deze twee algoritmen zijn slechts voorbeelden van de manier waarop domein decompositie kan worden opgezet. De manier

waarop de oplossingen van de deeldomeinen worden gekoppeld is in principe vrij te kiezen en er bestaan zeer veel varianten. In het algemeen probeert men een koppeling te bedenken die ervoor zorgt dat het iteratieve proces zo snel mogelijk convergeert. Er bestaan overigens ook varianten die niet iteratief te werk gaan.

In bovenstaande voorbeelden is domein decompositie beschreven op het niveau van continue (d.i. niet-gediscretiseerde) vergelijkingen. Op het niveau van gediscretiseerde vergelijkingen doen zich allerhande vragen voor met betrekking tot de manier waarop variabelen van het rekenrooster van het ene domein overgezet moeten worden naar variabelen op het andere rekenrooster. Bij overlappende domein decompositie moet men bijvoorbeeld de gediscretiseerde oplossing van deeldomein Ω_1 gebruiken als randvoorwaarde voor deeldomein Ω_2. Daarbij moet men uiteraard wel de gegevens uit de roosterpunten van deeldomein Ω_1 interpoleren naar de randpunten van het rooster van deeldomein Ω_2. De manier waarop deze interpolatie gebeurt, kan belangrijk zijn voor de kwaliteit van de uiteindelijke oplossing.

Domein decompositie wordt veelal gebruikt in gevallen waarbij het niet of nauwelijks mogelijk is om een gebied met een enkelvoudig rooster goed te beschrijven. Bij het ontwerpen van vliegtuigen is het bijvoorbeeld een zeer veel gebruikte techniek omdat de geometrie van vliegtuigen in combinatie met de fysica van luchtstroming rondom het vliegtuig het lastig maakt om met een enkelvoudig rooster te werken.

Binnen WAQUA/TRIWAQ wordt het vooral gebruikt om de omvang van het rekenwerk terug te brengen. Doorgaans zijn er in modellen bepaalde gebieden waar men met kleine maaswijdte of een groot aantal lagen wil rekenen om lokale effecten te bestuderen of omdat de fysica van het gebied om een dergelijk fijn rooster vraagt. Om nu te voorkomen dat de kleine maaswijdte of het grote aantal lagen ook gebruikt moet worden in stukken van het model die dat niet nodig hebben, kan men het model opdelen in deeldomeinen en per deeldomein het aantal lagen en de maaswijdte kiezen.

Merk op dat de opdeling in deeldomeinen vooral wordt bepaald door de geometrie en fysica van het model en de wensen van de gebruiker. Het doel is in elk geval niet om deeldomeinen van gelijke omvang te krijgen. Dit in tegenstelling tot parallel rekenen, waarbij de opdeling zich niets aantrekt van de fysica van het model en uitsluitend gericht is op het verkrijgen van deeldomeinen van gelijke omvang. Bij parallel rekenen kan de opdeling dan ook automatisch gemaakt worden terwijl bij domein decompositie het inzicht en de wensen van de gebruiker onmisbaar zijn.

4.2 Globaal ontwerp: DDVERT, DDHOR, toekomst

De manier waarop domein decompositie in SIMONA is gerealiseerd, verschilt enigszins tussen domein decompositie met horizontale verfijning en domein decompositie met verticale verfijning. Dit verschil is terug te voeren op het feit dat bij verticale verfijning in principe gebruik kan worden gemaakt van een enkele invoer file, omdat het rooster in horizontale zin een gewoon rekenrooster is. Bij horizontale verfijning moet per deeldomein een invoer file gemaakt worden, omdat het rekenrooster in horizontale zin niet meer in zijn geheel beschreven kan worden op de manier die SIMONA gebruikt.

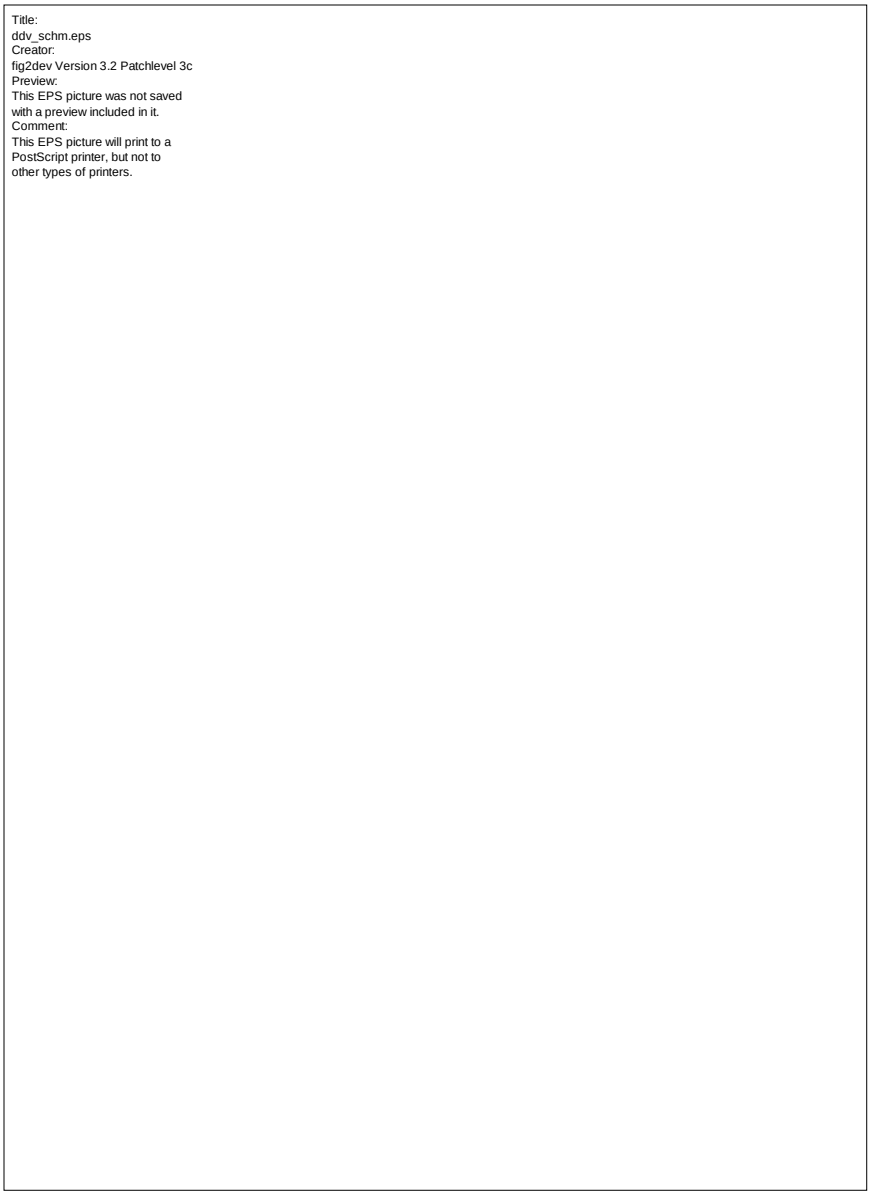
In Figuur 7 en Figuur 8 is weergegeven hoe domein decompositie in beide gevallen gerealiseerd is.

4.2.1 Domein decompositie met verticale verfijning

In Figuur 7 wordt getoond hoe domein decompositie met verticale verfijning te werk gaat. Er is een enkele invoer (siminp-)file die het gehele domein beschrijft. Deze invoerfile bevat niet rechtstreeks een waarde voor het aantal lagen: op de plek waar het aantal lagen gespecificeerd moet worden, staat het symbool ‘%KMAX%’. Bovendien is eventuele invoer die afhangt van het aantal lagen (bijvoorbeeld de laagverdeling) niet direct in de invoerfile gegeven, maar in aparte invoerfiles per deeldomein, die doormiddel van include-statements ingevoegd worden in de siminp-file.

Behalve de siminp-file en de aparte include-files per deeldomein moet de gebruiker nog een zogeheten areas-file toevoegen, waarin hij beschrijft hoe de opdeling in deeldomeinen is. Deze file kan zowel handmatig als met behulp van VISIPART gemaakt worden. Afgezien van het klaarzetten van alle invoer, hoeft de gebruiker verder niets te doen dan het aanroepen van het runscript.

Allereerst wordt voor elk voorkomend aantal lagen WAQPRE gedraaid, die een SDS-file maakt voor het betreffende aantal lagen voor het hele domein. Daaruit worden vervolgens met COPPRE die deeldomeinen gehaald die dat aantal lagen hebben. Stel dat men bijvoorbeeld 3 deeldomeinen heeft, waarvan de eerste twee allebei 2 lagen gebruiken en de laatste 4 lagen, dan wordt dus twee keer WAQPRE gedraaid (eenmaal voor een SDS-file met 2 lagen, waaruit COPPRE deeldomeinen 1 en 2 genereert, en eenmaal voor een SDS-file met 4 lagen, waaruit COPPRE deeldomein 3 genereert). Op deze manier ontstaat per deeldomein een SDS-file met het gewenste aantal lagen.



Figuur 7: globale opzet van een DDVERT berekening

COEXEC (zie de beschrijving van parallel TRIWAQ) zal voor elk deeldomein een TRIWAQ process opstarten. Dit process zal de berekeningen voor dit domein voor zijn rekening nemen. Randvoorwaarden worden door middel communicatie met de processen van de aangrenzende domeinen verkregen. Omdat de aangrenzende deeldomeinen in principe een ander aantal lagen hebben, zal bij die communicatie ook interpolatie plaats moeten vinden. Dit alles (communiceren en interpoleren) gebeurt door de communicatie bibliotheek COCLIB.

Na afloop van de berekeningen bevatten de SDS-files van de deeldomeinen de resultaten. Door COPPOS worden deze resultaten vervolgens samengevoegd tot een SDS-file waarin het aantal lagen gelijk is aan het maximaal aantal lagen dat in de berekening gebruikt is. Daarbij worden gegevens van deeldomeinen met kleinere aantallen lagen geïnterpoleerd naar het gewenste aantal lagen.

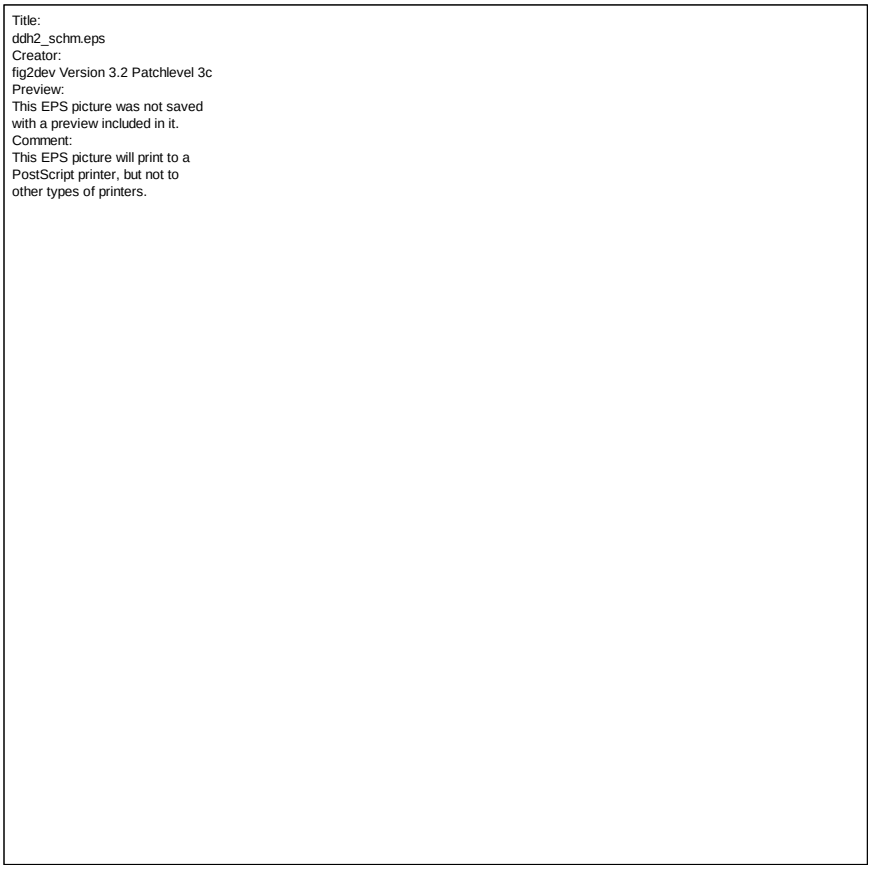
In bovenstaand voorbeeld (twee domeinen met 2 lagen en één met 4 lagen) zal de uiteindelijke file dus 4 lagen bevatten. Merk op dat de verticale resolutie van de gegevens in deze file dus gedeeltelijk schijn is: alleen in de deeldomeinen die daadwerkelijk met het maximaal aantal lagen zijn doorgerekend is de verticale resolutie correct. In overige deeldomeinen wordt de verticale resolutie door middel van interpolatie opgekrikt. Dit kan van belang zijn bij het interpreteren van de resultaten van een simulatie.

Een bijzondere situatie doet zich nog voor wanneer men een berekening met verticale verfijning wil herstarten. Men heeft immers alleen restartinformatie voor het maximaal aantal lagen dat gebruikt is. Om deze situatie aan te kunnen, heeft WAQPRE de mogelijkheid om door middel van interpolatie de restartinformatie uit een file met een groot aantal lagen te gebruiken voor het maken van een SDS-file met een kleiner aantal lagen.

4.2.2 Domein decompositie met horizontale verfijning

In Figuur 8 wordt getoond hoe een berekening met horizontale verfijning in zijn werk gaat. Nu levert de gebruiker meerdere invoerfiles met in elke invoerfile een rekenrooster voor (een gedeelte van) het domein. Door middel van een areas-file geeft de gebruiker per siminp-file aan welke stukken hij van het rekenrooster wil gebruiken en welke stukken niet.

Het is dus vrij gemakkelijk om een deel van een bestaand model lokaal te verfijnen. Men levert de siminp-file voor het bestaande model en door middel van de areas-file geeft men aan dat men het stuk dat verfijnd moet worden niet uit dit model wil gebruiken. Daarnaast levert men een tweede siminp-file waarin een rooster voor het verfijnde gebied is gedefinieerd. De areas-file behorend bij deze tweede invoerfile zal gewoonlijk aangeven dat men het gehele rooster wil gebruiken.



Figuur 8: globale opzet van een DDHOR berekening

Ook kan men meerdere deeldomeinen uit één domein halen: bijvoorbeeld bij een lang, recht kanaal kan men twee deeldomeinen uit één siminp file genereren: één voor het begin van het kanaal en één voor het eind. Het stuk daartussenin kan men dan met een andere siminp-file specificeren. Op die manier krijgt men drie deeldomeinen uit twee siminp-files.

Per siminp-file wordt nu het runscript waqpre.run gedraaid, dat ervoor zorgt dat er SDS-files voor alle deeldomeinen gegenereerd worden die uit die siminp-file gemaakt moeten worden. Dit gebeurt door WAQPRE te draaien om een SDS-file voor het hele domein te maken en daar door middel van COPPRE de deeldomein-SDS-file uit te genereren.

Als de gebruiker op die manier de SDS-files voor de deeldomeinen gegenereerd heeft, kan hij vervolgens de berekening opstarten. Merk op dat de gebruiker dus expliciet de SDS-files voor de deeldomeinen ziet, in tegenstelling tot wat er bij parallel rekenen en domein decompositie met verticale verfijning gebeurt.

De berekening zelf verloopt analoog aan die bij verticale verfijning: de TRIWAQ processen die door COEXEC worden opgestart communiceren met elkaar voor het uitwisselen van randvoorwaarden, waarbij ook weer interpolatie plaatsvindt (ditmaal horizontale interpolatie).

Na afloop van de berekening worden de resultaten door COPPOS samengevoegd tot SDS-files per siminp-file. Desgewenst kan men echter ook direct gebruik maken van de deeldomein SDS-files voor het interpreteren van de resultaten.

4.2.3 De toekomst

Op dit moment kan men beide vormen van domein decompositie combineren met parallel rekenen maar niet met elkaar. Deze beperking zal binnenkort weggenomen worden.

Verder is het op dit moment niet mogelijk om bijvoorbeeld in het ene domein wel transport te gebruiken en in het andere niet. Er wordt op dit moment volop gedacht om dergelijke mogelijkheden wel te gaan ondersteunen, maar dit zal niet op korte termijn gerealiseerd worden. Ook kan men nog niet de tijdstap per deeldomein apart kiezen: alle deeldomeinen moeten met dezelfde tijdstap rekenen. Vooral voor grove deeldomeinen is dat vaak jammer, omdat er in principe met een langere tijdstap gerekend zou kunnen worden, wat flink wat rekentijd zou kunnen schelen. Ook dit is een idee voor toekomstige ontwikkeling.

4.3 Interpolaties COCLIB, COPPOS, WAQPRE

De strategie die wordt gevolgd voor het implementeren van domein decompositie in WAQUA/TRIWAQ is ontstaan vanuit de strategie die wordt gebruikt voor parallel rekenen.

Die strategie voor parallel rekenen bestaat uit het opstarten van aparte rekenprocessen, een per deeldomein, ieder met een lokale hoeveelheid geheugen met daarin de gegevens van het eigen deeldomein. Rond ieder deeldomein wordt een “guard band” gemaakt, een gebied van drie extra rijen en kolommen van roosterpunten voor de opslag van kopieën van waardes (waterstanden, snelheden, ...) van naburige deeldomeinen. Per deeldomein worden de normale berekeningen uitgevoerd, inclusief alle gebruikelijke (hogere orde) differentiestencils, en waarbij er informatie nodig is van naburige deeldomeinen dan wordt die uit de guard band gelezen. Het up-to-date houden van de waardes in de guard band wordt door de programmeur gedaan door op geschikte momenten communicatie toe te voegen in het programma.

Gegeven deze strategie lag het voor de hand om voor domein decompositie met verticale verfijning als volgt te werk te gaan. Ieder deeldomein heeft lokaal geheugen met daarin de gegevens voor zijn eigen deeldomein, op een rooster met zijn eigen verticale resolutie (aantal lagen). De deeldomeinen zijn uitgebreid met een guard band met dezelfde verticale resolutie. De normale berekeningen worden uitgevoerd voor de roosterpunten van het eigen deeldomein, inclusief gebruik van de normale differentiestencils. Waarbij er wordt verwezen naar punten van naburige deeldomeinen dan worden de benodigde waardes uit de guard band gelezen. Deze komen nu echter niet meer één-op-één overeen met waardes uit het buurdomein, maar worden verkregen door middel van verticale interpolatie.

De benodigde interpolaties zijn toegevoegd aan de communicatieoperatie voor het updaten van waardes in de guard band. Er zijn verschillende interpolatiemethodes die op een groot aantal manieren kunnen worden toegepast. Ten eerste moet er worden gekeken naar de betekenis van de waardes die worden geïnterpoleerd: voor waardes die het gemiddelde per laag voorstellen moeten andere formules worden gebruikt dan voor waardes die een “sample” zijn in het midden van de laag. Ten tweede is de locatie van variabelen van belang: in de laagmiddens of op de laaginterfaces. Ten derde kunnen de interpolatiemethodes voor verschillende datastructuren worden toegepast, bijvoorbeeld zowel voor arrays voor het hele rooster (“fullbox” of “mnmaxk”) als voor arrays voor speciale punten (bijvoorbeeld controlestations). Ten slotte kunnen de interpolatiemethodes worden gevoed met verschillende coëfficiënten. In bepaalde gevallen wordt dit gebruikt om in iedere halve tijdstap van de simulatie nieuwe verticale posities van laaginterfaces door te geven aan de communicatiebibliotheek COCLIB.

De verticale interpolaties zijn vooralsnog alleen uitgeprogrammeerd voor de situatie dat het ene deeldomein strikt fijner is dan het andere, en niet voor het geval dat er voor sommige lagen verfijning is en voor andere vergroving. Verder is een eis dat de laaginterfaces uit het grove deeldomein in ieder roosterpunt en op ieder tijdstip continu aansluiten op laaginterfaces van het fijne deeldomein. Iedere laag van het grove deeldomein wordt dan verfijnd in een of meer lagen van het fijne deeldomein. De bijbehorende verfijningfactoren mogen verschillen per laag van het grove domein.

Het doorlopen van roosterlijnen zorgt ervoor dat de koppeling van lagen met een vaste dikte aan sigmalagen (ademend rooster) in veel gevallen is verboden. Het probleem daarmee is namelijk dat de laaginterfaces bij gebruik van sigmalagen meebewegen met de

waterstand, zodat niet is gegarandeerd dat steeds dezelfde lagen van het ene deeldomein zijn gekoppeld aan een laag van het andere deeldomein. Sigma-vast koppeling is alleen toegestaan bij koppeling van een deeldomein met een of meer vaste lagen aan een deeldomein met slechts een enkele laag (per definitie een sigmalaag). In dat geval zijn de laaginterfaces van het grove deeldomein namelijk het wateroppervlak en de bodem, waarvoor het continu doorlopen automatisch is gegarandeerd. Een belangrijk verschil in de implementatie van deze sigma-vast koppeling ten opzichte van de sigma-sigma/vast-vast koppeling is dat de interpolatiegewichten in het eerste geval tijdsafhankelijk zijn en in het tweede niet.

Vooralsnog zijn er alleen redelijk eenvoudige interpolatiemethodes uitgeprogrammeerd: constante interpolatie voor laaggemiddelde waardes en voor fluxen, en lineaire interpolatie voor puntwaardes. De constante interpolatie voor laaggemiddelden neemt in alle lagen van een fijn deeldomein de waarde van de corresponderende laag van het grove deeldomein en voor een laag van het grove deeldomein het gewogen gemiddelde van de corresponderende lagen van het fijne deeldomein. Voor fluxen komt er in een laag van het fijne deeldomein een fractie van de waarde van het grove deeldomein en in het grove deeldomein de som van de waardes van het fijne deeldomein. Voor puntwaardes in laaginterfaces wordt in het grove deeldomein de waarde van de corresponderende laaginterface van het fijne deeldomein genomen, in het fijne deeldomein wordt er voor sommige laaginterfaces een waarde geselecteerd, en voor andere laaginterfaces echte lineaire interpolatie toegepast.

De verticale interpolaties worden niet alleen gebruikt bij communicatie tussen de verschillende rekenprocessen per deeldomein, maar ook in de preprocessor WAQPRE en de collector COPPOS.

De collector COPPOS gebruikt verticale interpolatie om de resultaten per deeldomein te kunnen verzamelen in een enkele globale SDS-file die met de gebruikelijke postprocessing tools kan worden gevisualiseerd. Die globale SDS-file kan slechts een enkele verticale resolutie bevatten, hiervoor is gekozen voor de fijnste verticale resolutie van alle deeldomeinen. COPPOS past slechts zeer eenvoudige interpolatiemethodes toe: constante interpolatie voor laaggemiddelde waardes (d.w.z.: dezelfde waarde uit het grove deeldomein wordt gekopieerd naar verschillende overeenkomstige lagen van het fijne globale rooster) en lineaire interpolatie voor waardes op laaginterfaces.

De preprocessor WAQPRE gebruikt verticale interpolatie om herstarten van DDVERT-berekeningen mogelijk te maken. De

verticale interpolaties zijn nodig omdat WAQPRE een aantal keer wordt uitgevoerd, namelijk apart voor iedere verticale resolutie. Bij een herstart is daarbij steeds een globale (restart) SDS-file nodig met dezelfde verticale resolutie. Er is slechts een enkele globale SDS-file, namelijk die met de fijnste verticale resolutie. Verticale interpolatie wordt gebruikt voor het verminderen van het aantal lagen.

Een bijproduct van de verticale interpolaties in WAQPRE ten behoeve van DDVERT is dat het niet langer nodig is om in de restart-som met precies dezelfde laagverdeling en roosteropdeling te rekenen als in de oorspronkelijke som. Het is dus ook mogelijk om een sequentiële som met een ander aantal lagen door te starten. Dit komt doordat de oorspronkelijke laagverdelingen per deeldomein en de gebruikte roosteropdeling niet in de globale SDS-file zijn opgeslagen. De genoemde beperkingen zijn ook niet nodig omdat de algemene constante en lineaire interpolatieformules worden toegepast. Wel moeten de laagverdelingen aan dezelfde eisen voldoen als in het rekengedeelte (doorlopen van laaginterfaces).

In het geval dat wel precies dezelfde laagverdelingen en roosteropdeling worden gebruikt in de restart-som dan pakken de interpolatiemethodes als volgt uit. De lineaire interpolatie voor waardes op laaginterfaces reduceert tot het selecteren van de waardes voor de corresponderende laaginterface van het fijne globale domein, en de constante interpolatie voor laaggemiddelde waardes verwordt tot het berekenen van het gewogen gemiddelde van een aantal kopieën van dezelfde waarde. Hiermee is in principe een perfecte restart van de oorspronkelijke som mogelijk.

4.4 Matchen van Roosters

De strategie die wordt gebruikt voor domein decompositie met horizontale verfijning in WAQUA/TRIWAQ lijkt weer sterk op de strategie voor parallel rekenen en voor verticale verfijning. Per deeldomein wordt er een apart rekenproces opgestart met lokaal geheugen en een guard band, de normale discretisaties worden toegepast, en voor roosterpunten nabij deeldomeinranden gebruiken die dan geïnterpoleerde waardes uit de guard band.

Een aantal verschillen met verticale verfijning zijn echter als volgt:

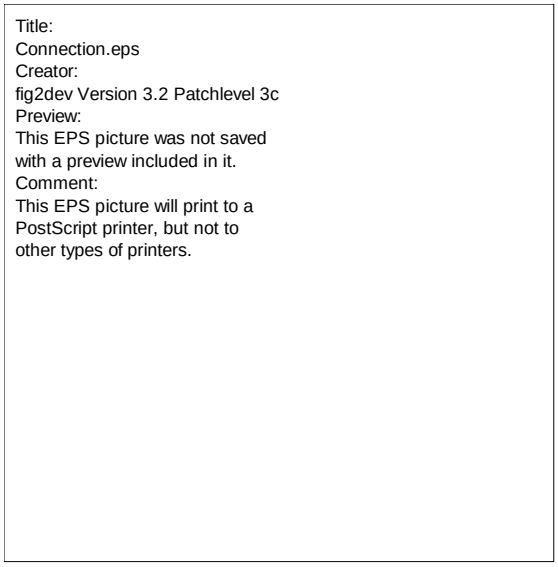
- Er zijn andere interpolatiemethodes nodig voor horizontale (2D) interpolatie dan voor verticale (1D) interpolatie, bijvoorbeeld bilineair of constant-per-cel voor de waterstanden en concentraties, en bilineair of “constant-per-cellface/lineair-haaks-erop” voor de stroomsnelheden.

- De implementatie van de interpolaties is flink anders, omdat er informatie van meerdere deeldomeinen nodig kan zijn voor een enkel interpolatiepunt en omdat die informatie niet kan worden opgeslagen in de datastructuren voor het eigen rooster.
- Het is niet meer mogelijk/gewenst om alle uitvoergegevens op een enkel rooster en in een enkele SDS-file te verzamelen.

Alleen het laatste punt is direct voor gebruikers van belang en wordt in deze paragraaf verder uitgewerkt. Vanwege dit punt is er namelijk voor gekozen om te werken met meerdere aparte roosters die aan elkaar gekoppeld worden, en die worden geïmplementeerd in aparte simulatie-invoerfiles (siminp-files).

De verschillende roosters worden ook wel “domeinen” genoemd. Het geheel van de gekoppelde roosters heet het “totale gebied”. Per domein kan een roosteropdeling in “deeldomeinen” worden gemaakt ten behoeve van parallel rekenen.

Per rooster/siminp-file kan de gebruiker een stuk uitschakelen, het zogenaamde “inactieve gedeelte van het domein”. Het overblijvende gedeelte kan dan zowel op openingen (open randen) als op subdomeinranden worden gekoppeld. De programmatuur bepaalt zelf hoe de verschillende deeldomeinen van de verschillende rekenprocessen op elkaar aansluiten. Daarbij moeten openingen ofwel volledig ofwel niet gekoppeld worden, en moeten subdomeinranden altijd volledig gekoppeld worden.



Figuur 9 : koppeling van roosters met horizontale verfijning op dieptepunten, hoekpunten van roostercellen

Figuur 10 : problematisch rooster voor horizontale verfijning. De rode roosterlijn is geheel zichtbaar in het linker domein en heeft daar twee verschillende m-coördinaten

De verschillende roosters worden aan elkaar gekoppeld op dieptepunten, d.w.z. op de hoekpunten van roostercellen, zie Figuur 9. Hiervoor neemt de programmatuur alle (*x*,*y*)-coördinaten van dieptepunten op openingen en deeldomeinranden en zoekt dan uit wat er aan elkaar past. Daarbij gelden er een aantal beperkingen die vergelijkbaar zijn met horizontale verfijning: er moet eenduidig sprake zijn van een grover en een fijner domein en roosterlijnen uit het grove domein moeten doorlopen in het fijne domein (samenvallen dieptepunten). De (geheeltallige) verfijningsfactoren worden door de programmatuur berekend en mogen variabel zijn langs de interface. In de richting haaks op de interface wordt een constante verfijningsfactor gebruikt, de programmatuur bepaalt de best passende waarde.

Naast bovenstaande eenvoudige beperkingen worden er nog een aantal andere eisen gesteld aan de te koppelen roosters. Ten eerste mogen er geen ξ-roosterlijnen (x-richting kromlijnig rooster) worden gekoppeld aan η-roosterlijnen ("y-richting") omdat hiermee U-snelheden in V-snelheden moeten worden omgezet. Ten tweede moet de oriëntatie (positieve ξ-richting) van de verschillende domeinen hetzelfde zijn. Verder moet iedere roosterlijn een begin en een einde hebben, cirkels zijn niet toegestaan. Ten slotte moeten deeldomeinen voldoende breed zijn voor de guard band van de buurdomeinen en moeten roosterlijnen die doorlopen binnen een enkel domein een unieke m- of n-

coördinaat hebben. Deze laatste eis wordt geïllustreerd in Figuur 10.

4.5 Nauwkeurigheidsaspecten

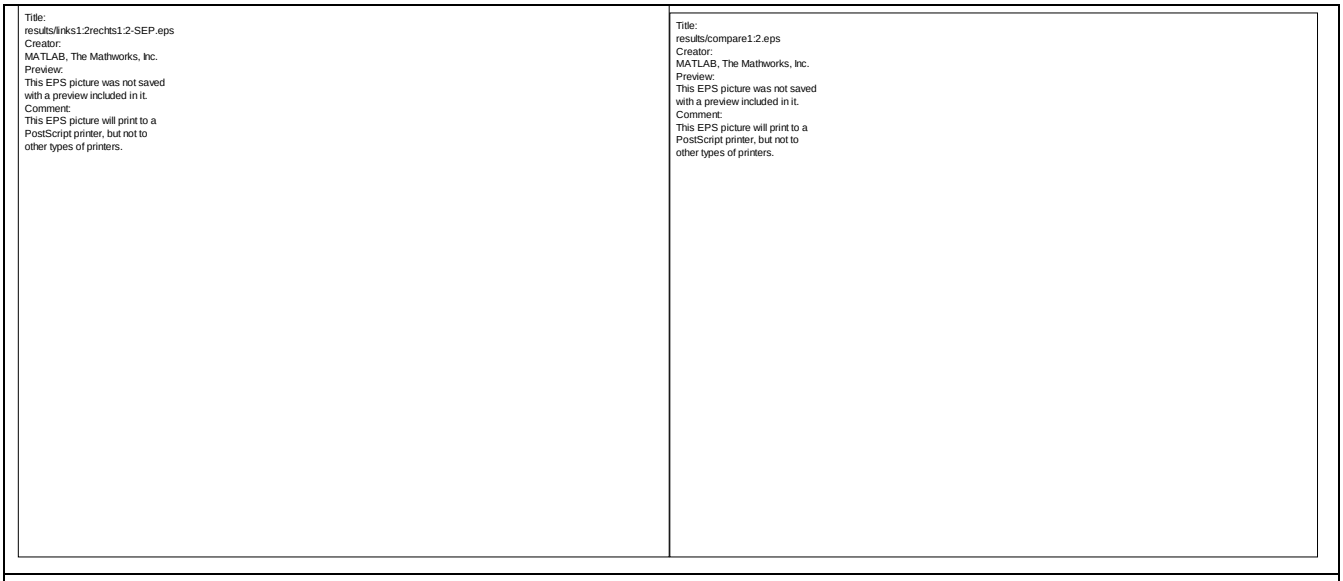
Bij de ontwikkeling van domein decompositie met verticale en horizontale verfijning zijn diverse testen uitgevoerd om de goede werking van de programmatuur te onderzoeken. Daarbij is nog niet heel gedetailleerd gekeken naar de effecten van verschillende koppelingstechnieken (m.n. interpolatiemethodes). Wel is ervoor gezorgd dat de met domein decompositie verkregen oplossingen globaal goed lijken op de oplossingen die worden verkregen zonder domein decompositie. Hiervoor zijn in sommige gevallen aanpassingen gemaakt aan de koppelingsvergelijkingen om bepaalde artefacten of instabiliteiten te vermijden.

De algemene ervaring met domein decompositie is dat de oplossing per deeldomein goed lijkt op de oplossing die wordt verkregen in een globale simulatie met dezelfde resolutie. Als bijvoorbeeld het Kuststrookmodel wordt gekoppeld aan het Zeedeltamodel, dan lijkt de oplossing in het actieve gedeelte van het Kuststrookmodel sterk op de oplossing die met het Kuststrookmodel wordt verkregen zonder horizontale verfijning. En de oplossing in het Zeedelta-gedeelte lijkt op oplossingen van het zelfstandige Zeedeltamodel met geschikte randvoorwaarden. Vergelijkbare ervaringen zijn opgedaan met verticale verfijning met het MOHA-model (monding Haringvliet).

In het algemeen kan dus worden gesteld dat het invloedsgebied van interfaces tussen verschillende (deel)domeinen beperkt is en dat de domein decompositie oplossing in ligt tussen de oplossingen voor de verschillende horizontale/verticale resoluties die er worden gebruikt.

Deze ervaringen worden geïllustreerd aan de hand van een eenvoudig testbakje dat is doorgerekend met horizontale verfijning, zie Figuur 11. Het bakje representeert een kanaal van 14,4 km lang met een diepte van 5 meter dat aan het rechter uiteinde is gesloten. Via de randvoorwaarde links wordt een lopende golf gecreëerd. In het linker gedeelte van het kanaal wordt een roosterafstand van 100 meter gebruikt, in het rechtergedeelte is de roosterafstand 200 meter.

De linker figuur toont de berekende waterstanden als functie van de tijd langs de middenas van het kanaal. In deze figuur zijn geen effecten van het gebruik van domein decompositie te zien behalve de witte lijn waardoor de oplossingen voor de twee verschillende domeinen van elkaar worden gescheiden.



Figuur 11 : resultaten voor een testbakje met horizontale verfijning. Links: waterstand in as van kanaal, rechts: verschil ten opzichte van onverfijnde berekeningen (zie tekst).

De rechter figuur toont het verschil ten opzichte van twee oplossingen die zijn berekend zonder domein decompositie. Voor het linker domein wordt de domein decompositieoplossing vergeleken met een som met overal een roosterafstand van 100 meter, in het rechter domein wordt vergeleken met een sequentiële som met overal 200 meter. Vanaf T=20min zijn er verschillen te zien die oplopen tot ongeveer 7mm. Deze grote verschillen komen echter doordat appels met peren worden vergeleken! De loopsnelheid van de golf is iets anders bij gebruik van roosterafstand 200 in plaats van 100 meter. Dus in de simulatie met overal 200 meter is de golf op een iets ander tijdstip bij de interface en dit veroorzaakt niet-relevante verschillen bij de vergelijking. De echte interessante verschillen zijn vanaf T=20min te zien in het linker domein. In de domein decompositieoplossing komen reflecties voor die in een simulatie met overal 100 meter niet bestaan. Deze reflecties zijn in dit geval ongeveer 1mm groot, op een waterstand van meer dan 1m.

5 Praktisch gebruik van Domein Decompositie en Parallel Rekenen

5.1 Aandachtspunten bij Parallel Rekenen

Bij het gebruik van parallel rekenen dient men er rekening mee te houden dat het meestal per berekening duurder (in geld) is dan sequentieel rekenen. Dit is omdat de efficiency (de versnelling per processor) van een parallelle berekening veelal kleiner is dan 1. Om hetzelfde aantal berekeningen te doen heeft men dus bij parallel rekenen meer processor-uren nodig dan bij sequentieel rekenen. De reden om toch gebruik te maken van parallel rekenen is dan ook voornamelijk om de doorlooptijd van een berekening te bekorten.

Als men bijvoorbeeld onafhankelijk van elkaar een twintigtal scenario's moet doorrekenen, kan men beter twintig keer een sequentiële som draaien (die dan wel eventueel naast elkaar op verschillende processoren van een multiprocessor kunnen draaien) dan twintig keer achter elkaar een parallelle som.

Als men echter twintig scenario's achter elkaar moet doorrekenen, waarbij de uitkomst van de ene mede bepalend is voor de invoer van de volgende, dan zal men juist wel van parallel rekenen gebruik willen maken. Of ook als de uitkomsten van een grote berekening om andere reden snel beschikbaar moeten zijn.

Overigens komt het wel voor dat een parallelle berekening een efficiency heeft die groter is dan 1 als gevolg van het beter gebruik van de cache. Dit is echter vaak op voorhand lastig te voorspellen en alleen als men een bekend model gebruikt op een aantal processoren waarop men al eerder voor dat model een hoge efficiency heeft gemeten, kan men hier op rekenen.

Verder dient men zich te realiseren dat parallel rekenen geen onbeperkte performancewinst oplevert. Er is voor ieder model en elk type computer een maximaal aantal processoren dat nuttig ingezet kan worden. Als meer dan dit aantal processoren inzet, dan levert dan nauwelijks nog extra performance op (maar wel rekenkosten).

Het maximaal aantal processoren dat gebruikt kan worden, wordt in technisch opzicht vooral bepaald door de verhouding tussen de tijd die een proces kwijt is met rekenen ten opzichte van de tijd die het aan communiceren besteed. Naarmate het aantal processoren stijgt wordt deze verhouding ongunstiger. Dit kan men zich eenvoudig voorstellen als men bedenkt dat bij een stripsgewijze verdeling van een model het aantal te communiceren items per processor even groot blijft, maar de grootte van het rekenwerk

omgekeerd evenredig is met het aantal processoren. Ook bij andere opdelingen kan men een soortgelijke redenering toepassen waaruit steeds blijkt dat processen verhoudingsgewijs steeds meer tijd gaan besteden aan communicatie.

Naarmate het model groter is, kunnen gewoonlijk meer processoren nuttig worden gebruikt omdat de hoeveelheid rekenwerk per proces langer groot blijft ten opzichte van de tijd die aan communicatie wordt besteed. Ook is het zo dat het maximaal aantal nuttige processoren voor TRIWAQ sommen met meerdere lagen groter is dan voor WAQUA sommen van vergelijkbare horizontale afmetingen.

Op computers met een hoge latency en/of een lage bandbreedte zal communicatie relatief veel tijd vergen en dus zal op dergelijke computers het maximaal aantal nuttige processoren voor een bepaald model kleiner zijn dan op een computer met een lage latency en een hoge bandbreedte.

Ook dient men te bedenken dat er altijd nog preprocessing (met WAQPRE) nodig is en vaak ook nog postprocessing, wat allemaal niet geparallelliseerd is. Het heeft op een gegeven moment weinig zin om een berekening nog verder te versnellen als de pre- en postprocessing toch al veel meer tijd kosten.

Een ander aspect waar men bij parallel rekenen mee te maken heeft is het feit dat het doorgaans plaats vindt op supercomputers of op computers waarop in elk geval meerdere gebruikers werken. In dat geval zijn er wachtrijsystemen om iedereen een eerlijke toegang tot de parallelle processoren te geven. In periodes van grote drukte kan het soms gebeuren dat een berekening langdurig in een wachtrij staat. Dit geldt des te meer als men voor een berekening een groot aantal processoren nodig heeft. Dit moet meegenomen worden in de beslissing om gebruik te gaan maken van parallel rekenen: het heeft geen zin om een berekening die een dag duurt met parallel rekenen te reduceren tot een berekening van een half uur, als men vervolgens een dag moet wachten voordat die berekening gedraaid kan worden.

Als laatste aandachtpunt bij het gebruik van parallel rekenen geldt nog dat het vaak toch enig inzicht van de gebruiker en beheerder vraagt. De gebruiker moet erop toezien dat de gebruikte partitie goed is (de automatisch door COPPRE gegenereerde partitie is redelijk maar vaak niet optimaal) en hij moet een goede keuze maken voor het te gebruiken aantal processoren. Ook is het werken op supercomputers soms wat lastiger dan op gewone computers (denk bijvoorbeeld aan het gebruik van het wachtrijsysteem). Voor de beheerder van WAQUA/TRIWAQ geldt dat het installeren van

de parallelle versie op een nieuwe parallelle computer vrijwel nooit helemaal triviaal is en soms ronduit lastig.

Al deze opmerkingen dienen goed afgewogen te worden als men gebruik wil gaan maken van parallel rekenen. Dat neemt niet weg dat in veel gevallen het gebruik van parallel rekenen mogelijkheden biedt die anders niet zouden bestaan.

5.2 Aandachtspunten bij Domein Decompositie met Verticale Verfijning

Domein decompositie met verticale verfijning is nadrukkelijk iets anders dan parallel rekenen. Waar men bij parallel rekenen min of meer hetzelfde aantal berekeningen doet (verdeeld over meerdere processoren) zal men verticale verfijning gewoonlijk gebruiken om het totaal aantal berekeningen dat nodig is voor een model te reduceren door een kleiner aantal lagen te gebruiken waar dat kan. Domein decompositie met verticale verfijning kan dus wel degelijk een kostenbesparing opleveren (naast een verkorting van de doorlooptijd). Ook hoeven de deeldomein in principe parallel doorgerekend te worden.

Daarnaast kan men domein decompositie met verticale verfijning gebruiken om 2D (WAQUA) functionaliteiten voor rivieren te combineren met modellen met meerdere lagen. Als men in een riviergebied slechts één laag gebruikt, kan men overlaten gebruiken, kan men de Nikuradse ruwheidsberekening toepassen en kan men de randvoorwaarde met automatische debietverdeling gebruiken. Dit terwijl men in hetzelfde model bij de kust gewoon 3D kan rekenen.

Verdere voordelen van domein decompositie zijn bijvoorbeeld dat men binnen één model verschillende ruwheidsformuleringen toepassen (bijvoorbeeld Manning in kustgebied, White-Colebrook in rivieren) en dat men minder numerieke problemen in rivieren heeft (denk aan zouttransport bij sterke bodemgradiënten). Daarnaast kan men bepaalde numerieke problemen bij open randen vermijden. Stel dat men bijvoorbeeld een grof model wil draaien om randvoorwaarden te genereren voor een ander model met een hoger aantal lagen, dan kan men in de buurt van de rand in het grove model het aantal lagen gebruiken waarvoor men de randvoorwaarden nodig heeft. De randvoorwaarden zullen dan beter aansluiten bij het fijnere model.

Deze voordelen betekenen echter wel dat de gebruiker zich er goed bewust moet zijn hoe hij domein decompositie gebruikt. Hij zal zelf de opdeling van het rooster moeten maken (of door een deskundige laten doen) op basis van de interessegebieden van de gebruiker en van de fysica van het model. In elk geval dient men

ervoor te zorgen dat er geen subdomeinranden liggen op plekken waar sterke gradiënten optreden (waar de gebruikte interpolaties eventueel moeite mee zouden kunnen hebben) of in de buurt van open randen en speciale constructies (zoals sluizen).

Zeker als men gebruik wil maken van parallel rekenen in combinatie met verticale verfijning (dus de deeldomeinen elk ook weer parallel laten doorrekenen) is een goed inzicht in de materie vereist en ook enige handigheid. Men zal dan per deeldomein een geschikt aantal processoren moeten kiezen en handmatig de roosteropdeling moeten balanceren. Het programma VISIPART kan hier veel ondersteuning bieden, maar kan het niet automatisch.

Bij het maken van een domein decompositie invoer is niet alleen de keuze van de deeldomein van belang, maar ook het gebruikte aantal lagen per deeldomein en de daarbij behorende overige fysische parameters. Zo moet men in gebieden met minder lagen bijvoorbeeld gewoonlijk een grotere diffusiecoëfficiënt gebruiken. Dit doet men door per subdomein en per fysische parameters aparte include files te maken en die vanuit de simulatie invoer file te includen met behulp van de daarvoor te gebruiken constructies (include <filename>.%KMAX%).

Ten slotte moet een gebruiker bij het interpreteren van de resultaten van een som met verticale verfijning zich goed realiseren dat de resultaten op de SDS-file in sommige stukken tot stand zijn gekomen door interpolatie vanaf een kleiner aantal lagen. Er is dus enige voorzichtigheid vereist bij het trekken van conclusies op basis van deze resultaten.

5.3 Werkplan Domein Decompositie met Verticale Verfijning

Bij het gebruik van domein decompositie met verticale verfijning kan men het volgende globale werkplan volgen.

- Ontwikkel een schematisatie met weinig lagen en zonder verfijning. Een dergelijke schematisatie is door een ervaren gebruiker relatief snel te maken en omdat het weinig lagen zijn kan er gemakkelijk mee getest worden.
- Bepaal een geschikte roosteropdeling en bijbehorende laagverdeling per deeldomein. Hierbij zal men primair uitgaan van de fysica van het model (waar moeten veel lagen gebruikt worden, waar weinig, waar treden gradiënten op) en van de eigen interesse (waar is hoge verticale resolutie nodig). Bovendien kan men rekening houden met een eventueel gebruik van parallel rekenen voor de deeldomeinen: dan kiest men een verdeling in deeldomeinen die later gemakkelijk gebalanceerd verder verdeeld kan worden.

- Pas de invoerfile aan voor domein decompositie met verticale verfijning. Dit is vooral een kwestie van het vervangen van het aantal lagen door %KMAX% en van laagafhankelijke informatie door include-files.
- Regel het model met verfijning af. Hierbij is het vaak zinvol om na te gaan hoe de resultaten van de verfijnde run zich verhouden tot die van een niet verfijnd model, waarbij men dan ook kan kijken wat er in het niet-verfijnde model verandert als men het aantal lagen verandert. Overigens is het niet altijd mogelijk om een run met een niet-verfijnd model te doen, bijvoorbeeld als men gebruik maakt van overlaten in een model dat verder 3D georiënteerd is.

5.4 Aandachtspunten bij Domein Decompositie met Horizontale Verfijning

De redenen om domein decompositie met horizontale verfijning toe te passen zijn in grote lijnen dezelfde als die voor het toepassen van verticale verfijning: het is mogelijk om de rekentijd te beperken door een fijn rooster alleen te gebruiken waar dat echt nodig is en men kan modellen maken die anders niet mogelijk waren geweest. Toch is het zo dat verticale verfijning vooral gebruikt zal worden door gebruikers die nu hun model gewoon sequentieel doorrekenen (en onnodig lang op het antwoord wachten) terwijl horizontale verfijning vooral gebruikt zal worden door gebruikers die zich nu bezig houden met het nesten/off-line koppelen van modellen.

Een voordeel van domein decompositie met horizontale verfijning ten opzichte van off-line koppelingen is dat de nauwkeurigheid van het fijnere model doorwerkt in het grovere model. Dit betekent ook dat als men een aanpassing doet in het fijnere model, dat het effect hiervan direct doorwerkt in het grove model. Waar men dus vroeger soms randvoorwaarden uit een grover model gebruikte voor meerdere, onderling enigszins verschillende fijne modellen, dan zal men nu betere voorspellingen krijgen.

Daar staat tegenover dat de on-line koppeling wel meer rekenwerk vergt. De terugkoppeling met het grovere model betekent een extra iteratieslag. Bovendien zal men nu de resultaten van een grover model niet hergebruiken voor meerdere runs met een fijner model. Verder is het zo dat men geen Kalman filtering meer kan gebruiken om de randvoorwaarden van het detailmodel te optimaliseren op basis van metingen.

Het gebruik van domein decompositie met horizontale verfijning vergt echter wel veel van de modelbouwer en de gebruiker. Per domein moet een aparte simulatie invoer file gemaakt worden,

waarvan de roosters goed op elkaar zijn afgestemd. Voor het verfijnen van speciale interessegebieden zal waarschijnlijk geschikte support komen waarmee men uit een bestaand rooster een deel kan selecteren en daar vervolgens (semi-)automatisch een fijner model voor kan laten genereren. In veel gevallen zal men echter toch modellen los van elkaar ontwikkelen. Daarbij moet men erop letten dat bepaalde aspecten van de procesformulering voor alle domeinen hetzelfde moet zijn (zoals de tijdstap, het al dan niet met transport rekenen). Als men ook nog per domein parallel rekenen wil gaan toepassen, dan kan het maken van een geschikte opdeling van de deelroosters een lastig karwei zijn dat veel kennis en ervaring vraagt.

De gebruiker is zich zeer bewust van het feit dat hij met domein decompositie met horizontale verfijning werkt. Hij zal per deeldomein WAQPRE moeten draaien en hij krijgt ook per deeldomein een uitvoerfile. Er is op dit moment vrijwel geen ondersteuning voor het gelijktijdig visualiseren van de resultaten uit de SDS-files van de verschillende domeinen. Daar staat uiteraard wel tegenover dat de resultaten in principe beter zijn dan bij off-line koppelingen en dat er modellen gebruikt kunnen worden die zonder domein decompositie niet mogelijk zouden zijn.