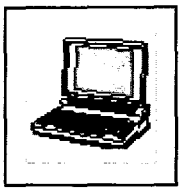


PROGRAMANT SUPERCOMPUTADORS

Jordi Garcia Almiñana

Estem vivint un moment històric en plena revolució tecnològica. Durant la década dels 40 i amb l'influència de la Segona Guerra Mundial, es van començar a desenvolupar els primers computadors orientats a executar de forma eficient aplicacions numériques relacionades amb càlculs militars. Actualment aquestes màquines es denominen supercomputadors, i el seu objectiu és executar a la màxima velocitat possible aplicacions que requereixen gran quantitat d'operacions numériques sobre estructures de dades molt grans. La supercomputació és un tema estratègic de recerca per a països com els Estats Units, el Japó, i darrerament també ho és a Europa.

L'objectiu d'aquesta col.laboració no és, ni molt menys, donar un curs complert ni tampoc incomplert de com es programa un supercomputador. El que es pretén és il.lustrar el tipus de problemes amb els que un programador es pot trobar a l'hora de programar una d'aquestes màquines, i descriure quina direcció segueix la recerca actual dins d'aquest camp per a tractar els problemes descrits.

L'estructura de la resta de l'article és la següent. Inicialment es descriu l'evolució dels supercomputadors provocada per la gran demanda de potència de càlcul en les aplicacions actuals. Es dóna una idea dels diferents tipus d'arquitectura, i quines limitacions té cada un. A continuació s'introdueix, mitjancant exemples, la problemàtica de programar aquest tipus de màquines, amb especial interès en els sistemes que ofereixen la possibilitat de paral.lelitzar. Finalment es dóna una visió de quin tipus d'eines automàtiques es necessita per donar suport al programador, i quin és l'estat actual en relació a aquestes necessitats.

JORDI GARCIA ALMIÑANA és professor del Departament d'Arquitectura de computadors i sistemes operatius a l'ETSETB

1. Arquitectura dels Supercomputadors

Avui en dia els supercomputadors executen models matemàtics que reemplacen els experiments físics. Els túnels de vent es converteixen en simulació aerodinàmica, el disseny de cotxes es basa en computadors gràfics per comptes d'en prototipus. Els models basats en computadors són el nucli de recerca actual sobre modelatge climatològic, tecnologia genètica o enginyeria aeroespacial, per citar alguns exemples. L'increment en la potència dels supercomputadors es reflexa en dos sentits. Per una banda hi ha nous problemes que es posen a l'abast per a poder ser modelats en els computadors. Per una altre banda permeten trobar millors solucions a problemes que ja eren coneguts. Per exemple, problemes en dues dimensions poden ser reformulats en tres dimensions. Aquest avanç és el resultat de la unió estreta de diferents àrees de recerca, tals com aplicacions, models numèrics, llenguatges d'alt nivell, compiladors, sistemes operatius, arquitectura, i disseny de la circuiteria bàsica.

Els models basats en computadors són el nucli de recerca actual sobre modelatge climatològic, tecnologia genètica o enginyeria aeroespacial, per citar alguns exemples.

La tecnologia que s'utilitza per fabricar els processadors que apareixen en les màquines d'altes prestacions, però, està arribant al límit físic de la velocitat de la llum. Encara que els components d'un processador assolissin aquesta velocitat, no es podrien executar més que alguns milions d'instruccions per segon. Per poder aconseguir millores significants en aquesta velocitat, els dissenyadors de sistemes centren els seus esforços en l'arquitectura. Bàsicament es coneixen dues tècniques per a aquesta finalitat: la *segmentació* i el *paral.lelisme*. La segmentació consisteix en descomposar les operacions en un nombre N d'etapes, cada una de les quals s'executa en un cicle de rellotge. Quan una instrucció passa a executar una etapa determinada, la següent instrucció ja pot començar a executar l'anterior, i així per totes les altres etapes. D'aquesta forma, en cada instant hi pot haver N instruccions executant-se simultàniament, i en cada cicle finalitza l'execució una instrucció. Els processadors que incorporen aquesta tècnica s'anomenen

processadors escalars segmentats. Malauradament, el nombre de fases en les que es pot descomposar una instrucció està molt limitat. Una variant de la segmentació és la vectorització, que consisteix en afegir al llenguatge màquina del processador instruccions que operen sobre vectors (instruccions vectorials), donant nom als *processadors vectorials*. La vectorització consisteix en executar la mateixa operació sobre cada un dels elements d'un vector a una freqüència d'una operació per cicle, reduint així la càrrega d'actualitzar els comptadors i punters a cada iteració del bucle.

Si es vol explotar el potencial d'aquests tipus de sistemes, els programes s'han d'escriure de forma que aprofitin o utilitzin al màxim cada un dels aspectes particulars de cada arquitectura.

La següent tècnica que permet millorar la velocitat d'un supercomputador es coneix com a paral.lelisme, i es basa en la rèplica de les unitats funcionals del processador, o en la rèplica dels processadors del sistema. Totes les unitats funcionals o els processadors col.laboren, preferiblement a parts iguals, en l'execució de l'aplicació. Segons aquesta classificació i com a exemple de processadors amb vàries unitats funcionals, es pot distingir entre els *processadors superescalars* i els *processadors Very Long Instruction Word*. Els primers poden iniciar l'execució de vàries instruccions per cicle, sempre i quan es detecti que les operacions es poden executar de forma independent. Per altre banda els processadors VLIW tenen instruccions llargues formades per un nombre fixe d'instruccions senzilles, les quals són empaquetades per un compilador.

El baix cost que assoleixen els microprocessadors descrits anteriorment, permeten la construcció de màquines amb múltiples processadors. Es pot distingir entre els *processadors en array*, els sistemes *multiprocessadors amb memòria compartida*, i els sistemes *multiprocessadors amb memòria distribuida*. Els processadors en array estan orientats a l'execució eficient d'operacions sobre vectors, i consten d'un gran nombre de processadors senzills que executen una mateixa operació sobre diferents elements d'un vector. Aquests processadors estan conectats a un processador escalar de propòsit general que controla l'execució de l'aplicació en tot el sistema. Els sistemes multiprocessadors amb memòria compartida estan formats per un conjunt de proces-sadors molt ràpids, els quals accedeixen a la mateixa memòria a través d'una xarxa d'inter-connexió. En aquest tipus d'arquitectura, els processadors poden treballar de forma conjunta per a resoldre l'aplicació més ràpidament, o cada un d'ells pot executar el seu propi programa. Per motius tecnològics, el nombre de processadors que poden compartir la memòria no pot ser gaire elevat (fins a 16 o 32), ja que resulta difícil que una sola memòria pugui subministrar dades a un nombre elevat de processadors a una velocitat raonable. És per aquest motiu que prenen importància els sistemes multiprocessadors amb memòria distribuida, també coneguts com a *processadors massivament paral.lels*, els quals poden arribar a conectar milers de processadors, cada un dels quals amb la seva pròpia memòria. A més, es poden utilitzar els processadors més avançats del moment sense necessitat de fer dissenys específics i cars, i el nombre de processadors del sistema es pot escalar fàcilment en funció de les necessitats de l'usuari. Per a que un grup de processadors col.labori en l'execució d'una aplicació, els processadors intercanvien informació mitjançant una xarxa d'interconexió a través de la qual envien i reben missatges. Normalment, però, el temps d'espera en rebre un missatge és una o vàries ordres de magnitud més elevat que el temps d'accés a la pròpia memòria. Per tant, tot i el potencial teòric que pot arribar a tenir aquest model d'arquitectura, és una qüestió clau el saber distribuir correctament el càlcul i les dades entre les memòries dels processadors, de forma que tots executin aproximadament el mateix nombre d'operacions, i que les dades requerides per aquestes es trobin a la memòria local de cada processador.

Una descripció molt més detallada de tots aquests sistemes es pot trobar a [1].

2. Programació de Màquines Paral.leles

Si es vol explotar el potencial d'aquests tipus de sistemes, els programes s'han d'escriure de forma que aprofitin o utilitzin al màxim cada un dels aspectes particulars de cada arquitectura. Normalment, els llenguatges de programació per a supercomputadors ofereixen al programador sentències o directives que permeten especificar com s'ha d'executar el programa per a que aquest sigui més eficient. Així dons, quan s'escriu un programa per a un sistema multiprocessador amb memòria compartida, s'ha d'estructurar el programa de forma que les diferents operacions que s'han d'executar es puguin repartir fàcilment per el conjunt de processadors del sistema, i de forma que cada un d'ells pugui realitzar el seu càlcul amb paral.lel amb els altres processadors. Si el multiprocessador té la memòria distribuida, a més del càlcul, també s'han de distribuir les dades entre les memòries de cada processador. L'objectiu d'aquesta secció és donar una visió general que quins aspectes s'han de tenir en compte al escriure un programa per a una màquina paral.lela.

2.1. Distribució del càlcul

Hi ha vàries formes de descomposar el càlcul d'un programa científic. Per una banda es podria partir el programa per les diferents tasques que realitza, cada una de les quals formada per un conjunt de bucles, de forma que cada una d'elles s'assignaria a cada processador. El problema és que de vegades una

aplicació no té un nombre suficientment gran de tasques per a poder assignar treball a tots els processadors, o que les tasques no són independents del tot, de forma que quan un processador treballa, els demés s'han d'esperar a que aquest acabi per a poder començar.

Per altre banda, les aplicacions científiques acostumen a utilitzar estructures de dades molt grans, les quals han de ser tractades repetidament dins de bucles. En aquest tipus d'aplicacions, el bucles representen la part computacional més important del programa. El paral.lelisme de dades consisteix en partir els bucles dels programes de forma que cada processador executi un conjunt de les iteracions. Per exemple, el següent fragment de codi:

Exemple 1:

```
do i = 1, 40
 do j = 1, 40
        A(i, j) = A(i, j) + 1
 enddo
enddo
```

podria ser paral.lelitzat assignant les iteracions del bucle i a diferents processadors. En aquest cas els processadors aniran recorrent simultàniament diferents files de la matriu A. El codi que expresa aquesta estrategia de paral.lelització serà del tipus:

```
forall i = 1, 40
 do j = 1, 40
        A(i, j) = A(i, j) + 1
 enddo
endforall
```

Però no sempre es pot paral.lelitzar el bucle que es vol, i de vegades inclús no es pot paral.lelitzar cap bucle sense transformar el codi. Per exemple, en el següent fragment de codi:

Exemple 2:

```
do i = 2, 40
 do j = 1, 40
        A(i, j) = A(i - 1, j) + 1
 enddo
enddo
```

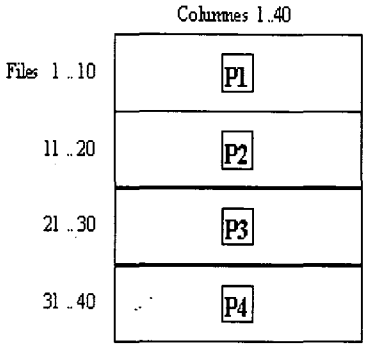
no es pot paral.lelitzar el bucle i, ja que per actualitzar la fila i es necessita la fila anterior i-1. És a dir, per a que el resultat de l'execució sigui correcte, no es pot actualitzar una fila si abans no ha estat actualitzada la fila anterior. En aquest cas es diu que hi ha una dependència en el bucle i, i per tant aquest bucle s'ha d'executar respectant l'ordre sequencial. En canvi si que es pot paral.lelitzar el bucle j. En aquest cas, per a cada fila i de la matriu A, cada processador podria recòrrer en paral.lel diferents parts d'aquesta fila.

Quan tots els bucles tenen dependències hi ha altres tècniques que, transformant el codi, permeten extreure certa paral.lelització. La descripció d'algunes d'aquestes tècniques es pot trobar a [2].

2.2. Distribució de les dades

Si el sistema multiprocessador per al qual s'escriu el programa té la memòria distribuida, una decisió important que s'ha de prendre és a on s'emplacen inicialment les dades. Aquesta qüestió està estretament relacionada amb la paral.lelització del càlcul, ja que en funció de la porció de l'estructura de dades que tingui un processador, serà més adequat paral.lelitzar un bucle o un altre. Generalment aquest tipus de programes segueixen la owner computes rule, és a dir, el processador propietari de la dada que s'ha de modificar és el responsable d'efectuar el càlcul. Si un processador té una dada que necessita un altre processador per a efectuar el seu càlcul, aquests s'han de comunicar mitjancant un missatge enviat per el processador que té la dada cap al processador que ha de fer el càlcul. El temps que tarda aquest missatge en arribar al seu destí és molt més elevat que el temps d'accés a memòria, per tant també s'ha d'intentar minimitzar el nombre d'accessos remots.

Per exemple, assumint que hi ha 4 processadors al sistema, distribuir la matriu A per files significa que cada processador té 10 de les files de la matriu tal i com mostra la següent figura. A nivell de repartició del càlcul i respectant el owner computes rule, significa que el primer processador és el responsable de recòrrer les 10 primeres files de la matriu, el segon processador ho és de les 10 següents files, i així successivament.



Tornant a considerar el codi de l'exemple 2 i assumint que A està distribuida per files, significa que s'ha de particionar el bucle i de forma que cada processador només recorri les 10 files que li correspon. Degut a la dependència en aquest bucle, cap processador no pot començar fins que el processador anterior hagi acabat les seves 10 files. A més, per a que el processador 2 pugui començar a modificar la fila 11, necessita la fila 10, que es troba a la memòria del processador 1. Per tant el bucle s'acabaria executant seqüencialment, amb el retard adicional d'haver d'esperar a rebre una fila sencera del processador anterior. Evidentment en aquest cas la millor alternativa seria partir la matriu A per columnes, de forma que cada processador pugui actualitzar paral.lelament les seves 10 columnes.

Hi ha molts altres aspectes que s'han de tenir en compte al distribuir les dades. Per exemple, segons el següent fragment de codi:

Exemple 3:

```
do i = 2, 40
 do j = 1, 40
        A(i, j) = B(j, i) + 1
 enddo
enddo
```

la matriu A es pot distribuir tant per files com per columnes.

Però un cop decidit com estarà distribuida la matriu A, per exemple per files, seria beneficiós adonar-se'n que la matriu B hauria d'estar trasposada respecte a la matriu A, és a dir, per columnes. Si no és així, el codi es podria executar igualment en paral.lel, però hi haurà un cost molt elevat en quant a comunicació, ja que la major part de les dades requerides per un processador per a efectuar el càlcul estan en la memòria d'un altre processador.

3. Eines de Suport a la Programació

Hi ha diferents alternatives per aplicar les tècniques descrites en la secció anterior, en funció del grau de coneixement que el programador d'aplicacions té del seu supercomputador. Per una banda podria ser el mateix programador qui decidís l'estratègia de paral.lelització, usant un llenguatge amb construccions que permetin expresar paral.lelisme. Aquesta tasca, però, no acostuma a ser gens fàcil i és propensa als errors. A més, existeixen milions de linies de codi d'aplicacions que inicialment van ser escrites pensant en computadors seqüencials, que haurien de ser adaptades a les característiques del supercomputador. Una segona alternativa és deixar que sigui el compilador de la màquina qui s'encarregui de detectar i explotar el possible paral.lelisme de l'aplicació. En aquest cas es faria automàticament una reestructuració del codi seqüencial generant un nou codi paral.lel optimitzat per la màquina en qüestió [2]. Actualment, tots els fabricants de supercomputadors ofereixen, junt amb la màquina, el propi compilador optimitzador.

Si a més, el supercomputador té la memòria distribuida, el programador ha de controlar les referències a dades no locals, insertant oportunament les sèntencies de pas de missatges. La gestió de la comunicació a aquest nivell és molt delicada, els programes resultants són difícils de depurar, i dificulta la portabilitat del codi a altres arquitectures. L'estil de programació resultant es pot comparar a la programació en assemblador d'una màquina seqüencial [3].

Els llenguatges amb paral.lelisme de dades ofereixen al programador la possibilitat d'escriure els programes assumint que les dades són totes globals. El programador només ha d'especificar, mitjancant directives, l'emplaçament de les dades, i el compilador s'encarrega de paral.lelitzar els bucles corresponents i de generar les sentencies de pas de missatges apropiades. D'aquesta forma, el mateix codi serveix per a multiprocessadors amb memòria compartida i per a multiprocessadors amb memòria distribuida. Actualment s'està treballant en definir un estàndard per aquest tipus de llenguatges [4], i comencen a haver compiladors que porten a terme aquest procés, insertant les corresponents crides de pas de missatge al codi inicial segons la distribució de dades especificada.

Els llenguatges amb paral.lelisme de dades ofereixen al programador la possibilitat d'escriure els programes assumint que les dades són totes globals.

Encara s'ha de portar a terme molta més recerca en aquest camp per a que els compiladors siguin realment eficients. Malgrat això, el programador és encara qui ha d'especificar l'emplaçament de les dades. Aquesta decisió és molt important, ja que fixa el paral.lelisme de l'aplicació i determina la quantitat d'accessos remots que s'han de portar a terme durant l'execució del programa. La distribució òptima de les dades depen de l'estructura del programa, del comportament del compilador, i de les característiques de la màquina. Les eines de distribució automàtica de dades [5][6] ajuden al programador en aquesta tasca. A partir de l'anàlisi del codi i tenint en compte certes característiques de la màquina, estimen de forma conjunta l'impacte que tenen les possibles distribucions de dades en quant a paral.lelisme i comunicació. Segons aquest anàlisi, inserten al codi original les corresponents directives de distribució de dades, el qual es compila amb algun dels compiladors descrits anteriorment.

L'èxit final que puguin tenir aquests tipus d'arquitectures dependrà, en bona part, dels resultats que donin les eines automàtiques. La potència de càlcul és avui en dia una realitat, però no servirà de res si no hi ha mitjans per poder aprofitar-la.

Referències

[1] J. L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Second edition. Morgan Kaufmann publishers, San Francisco, CA, 1996.

[2] M. Wolfe. *Optimizing Supercompilers for Supercomputers.* Research monographs in parallel and distributed computing. The MIT Press, Cambridge, MA, 1989.

[3] P. J. Hatcher, M. J. Quinn. *Data-Parallel Programming on MIMD Computers.* The MIT Press, Cambridge, MA, 1993.

[4] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele, M. E. Zosel. *The High Performance Fortran Handbook.* Scientific and engineering computation series. The MIT Press, Cambridge, MA, 1994.

[5] E. Ayguadé, J. Garcia, M. Gironès, M. L. Grande, J. Labarta. *DDT: A Research Tool for Automatic data Distribution in HPF.* Published in Scientific Programming, special issue on HPF, 1996.

[6] J. Garcia, E. Ayguadé, J. Labarta. *Dynamic Data Distribution with Control Flow Analysis.* In proceedings of Supercomputing'96 conference. Pittsburgh, PA, 1996.