Tietojenkäsittelytiede **28** Joulukuu 2008 sivut 55–65
Toimittaja: Jorma Tarhio

Rinnakkaistietokoneen uusi tuleminen

Martti Forsell
Valtion Teknillinen Tutkimuskeskus
Martti.Forsell@vtt.fi

Ville Leppänen
Turun yliopisto, Informaatioteknologian laitos
Ville.Leppanen@it.utu.fi

Martti Penttonen
Kuopion yliopisto, Tietojenkäsittelytieteen laitos
penttonen@cs.uku.fi

Tiivistelmä

Olemme tottuneet “Mooren lain” mukaiseen prosessorien tehon kasvuun. Viime vuosina nopeuden kasvu on kuitenkin alkanut taittua, ja sen sijaan prosessorivalmistajat ovat alkaneet rakentaa mikropiirille useampia prosessoriytimiä, joita ei kuitenkaan toistaiseksi osata käyttää hyväksi kovin tehokkaasti.

Rinnakkaislaskenta on ollut suuri lupaus jo monen kymmenen vuoden ajan, mutta sen käyttö on marginaalista lukuunottamatta joitakin tärkeitä tieteellisiä sovelluksia. Syy tilanteeseen on se, etteivät laitetekniikka ja algoritmiikka ole kohdanneet. Tarkastelemme uutta tapaa rakentaa prosessori niin, että prosessoriytimet olisivat käytettävissä mahdollisimman helppoa ohjelmointimallia käyttäen.

1 Mitä Moore sanoi

Useimmat tietotekniikan kehitystä seuraavat ovat kuulleet “Mooren laista”, jonka mukaan tietokoneen teho kaksinkertaistuu noin vuodessa ja joka laki näyttää luonnonlakejakin uhmaten pätevän vuosikymmenestä toiseen. Eihän sellaista lakia oikeasti ole. Vuonna 1965 G. Moore julkaisi artikkelin [17], jossa hän tarkasteli mikropiirille pakattujen komponenttien määrän kasvua viimeisen viiden vuoden aikana ja arvioi, kuinka kehitys voisi jatkua seuraavan kymmenen vuoden aikana. Vuonna 1959 lähdettiin yhdestä komponentista, vuonna 1965 voitiin pakata 50 komponenttia piirille ja ennusteen mukaan vuonna 1975 ehkä olisi mahdollista pakata minimikustannuksin 65000 komponenttia piirille. Toisin sanoen 16 vuodessa pakkaustiheys kasvaisi 2^{16}-kertaiseksi. Vuonna 2007 pakkaustiheys ei kuitenkaan ollut 2^{48} vaan likimain 2^{32} eli alle 5 miljardia. No, “laki” sittenkin piti paikkansa hiukan lievennettynä: “pakkaustiheys tuplaantuu puolessatoista vuodessa”. Sa-

OHJE KIRJAPAINOLLE: B5-arkin vasen- ja yläreuna kohdistetaan A4-arkin vasempaan ja yläreunaan. Näin pitäisi marginaaliksi tulla taitteen puolella noin 33 mm ja muualla noin 22 mm.

malla ovat myös prosessorien kellotaajuudet nousseet (nykyisin 4 GHz:iin saakka) ja tiedonsiirron kaistanleveydet kasvaneet (luokkaa 10 Tb/s) ja niillekin on esitetty vastaavia “lakeja”.

Vaikka Mooren lain kaltainen keksitty “laki” on yllättävän pitkään näyttänyt uhmaavan luonnon lakeja, on selvää, ettei sama kehitys voi jatkua loputtomasti, sillä eihän transistori voi tulla atomia pienemmäksi. Pakkaustiheys voi vielä jonkin aikaa kasvaa, mutta ennen pitkää pakkaustiheyttä kasvatettaessa johtojen ohuus johtaa satunnaisilmiöihin. Nykyisin käytetyssä 45 nm:n tekniikassakin (joka vastaa parinsadan pii-atomin kerrosta) jotkin transistorin osat ovat huomattavasti 45 nm:ä ohuempia! Toinen suuri ongelma, suuresta pakkaustiheydestä ja korkeasta kellotaajuudesta johtuva lämmöntuotantoongelma, on jo tullut vastaan — mikropiiri sulaa, jos sen läpi johdetaan liian suuri virta. Kun otetaan huomioon, että piirin käyttöjännitettä täytyy periaatteessa kasvattaa taajuutta nostettaessa, lämmöntuontanto kasvaa kuutiollisesti kellotaajuuden kasvaessa. Kolmas laskennan nopeutta rajoittava tekijä on tiedonsaannin hitaus. Kellotaajuuden ollessa 3 GHz sähkösignaali etenee syklin aikana 45 nm:n piirillä vain parisen milliä kapasitiivisten ja resistiivisten efektien takia, joten prosessori joutuu odottamaan tiedon saantia muistista yhä pitempään. Myös komponenteilla on oma vastaava hitautensa. Fyysisten rajoitteiden ohella suuri, ehkä jopa suurin, laskentatehoa rajoittava ongelma on tietojenkäsittelytieteellinen: miten käyttää parhaalla mahdollisella tavalla hyväksi prosessorin pinta, ts. sille pakatut komponentit niin, että prosessorista saataisiin laskennan kannalta paras hyöty.

Prosessoriteollisuus on todella kohdannut nämä ongelmat niin, että on käytetty jopa ilmausta: “prosessoriteollisuus on paniikissa” [21]. Eikö kuluttaja saakaan ensi vuonna kaksi kertaa tehokkaampaa tietokonetta? Lisätehoa ei enää saadakaan täysimittaisesti lisäämällä prosessoriin aputoimintoja ja korottamalla kellotaajuutta. Sen sijaan prosessoripiirille on alettu rakentaa useampia prosessoriytimiä toivoen, että tietokoneen tehoa voidaan moninkertaistaa moninkertaistamalla prosessoriytimien määrä. Paniikkiin on aihetta, sillä sovelluksen hyödynnettävissä oleva laskennallinen teho ei automaattisesti kasva prosessorien yhteenlasketun tehon mukana, jos lainkaan. Sama skaalautuvuusongelma koskee klusteriperiaatteella rakennettuja supertietokoneita, jotka on rakennettu suuresta määrästä nopeita prosessoreja: rinnakkaisuus hyödyttää vain, jos tehtävä voidaan riittävän karkealla tasolla jakaa melko riippumattomiin rinnakkain suoritettaviin osiin. Rinnakkaislaskenta on syytä ottaa uudelleen tarkasteluun, sillä rinnakkaistietokoneet ovat tulleet ajankohtaisemmiksi kuin koskaan.

2 Osattiin ennenkin rakentaa rinnakkaistietokoneita

Tietokoneen ensi vuosikymmeninä rinnakkaistietokoneet olivat suhteellisesti yleisempiä kuin viime vuosikymmeninä. Esimerkiksi neljäkymmentä vuotta sitten rakennettu ILLIAC IV oli yksi aikakautensa merkittävimmistä tietokoneista ja nimenomaan rinnakkaistietokone. Sen sanotaan olleen vuoteen 1981 saakka nopein tietokone [2], jolloin sen tittelin peri CRAY. ILLIAC IV:ssä oli 64 prosessointiyksikköä, jotka oli yhdistetty ristikkoverkolla. Prosessorin kellotaajuus oli n. 4 MHz, ja sillä oli 2048 sanaa 64-bittistä muistia. Massamuistin saantiaika oli 20 ms ja tiedonsiirtonopeus yhtei-

sestä massamuistista prosessorin keskusmuistiin oli 5×10^8 b/s. Nopeustittelin vuonna 1981 perinyt CRAY-1 oli vektoriprosessoria käyttävä rinnakkaistietokone. Näissä koneissa prosessorin laskentayksikkö on kloonattu vektoriksi, jossa osat suorittavat samanlaisia tehtäviä toisistaan riippumattomasti tai liukuhihnaperiaatteella. Erityistä huomiota on kiinnitetty siihen, että tieto päämuistista saadaan nopeasti. CRAY-1:ssä tiedonsiirtonopeus oli 5×10^9 b/s. CRAY-1:n tehoksi arvioitiin 160 MFLOPS. CRAY:n vektoriprosessorien valmistus loppui pari vuotta sitten. Viimeisimmän mallin CRAY X1E:n nopeus oli 147 TFLOPS, ts. noin miljoonakertainen CRAY-1:een verrattuna.

Parin viime vuosikymmenen aikana mikroprosessorien kehitys on ollut nopeaa. Mikrotietokoneiden suuren volyymin vuoksi prosessoriteollisuudella oli rahaa juuri sellaisten prosessorien kehittämiseen ja standardiprosessoreista alkoi saada suoritustehoa halvimmalla. Alkoi massiivisesti rinnakkaisten hajautettua muistia käyttävien rinnakkaistietokoneiden aikakausi. Tämän kehityksen suuntaa näytti Caltech Cosmic Cube vuodelta 1985, jossa oli 64 Intel 8086/87 -prosessoria ja joka prosessorilla oli 128 KB muistia. Prosessorit oli kytketty toisiinsa 6-ulotteisella hyperkuutiolla. Yhteistä muistia ei ollut lainkaan, vaan ohjelmointi perustui viestinvälitysmalliin. Cosmic Cube oli sen aikakauden halpa “minisupertietokone”, jonka teho oli luokkaa 3 MFLOPS. Sen nykyaikaisessa perillisessä, IBM Roadrunner -superkoneessa on 122,400 prosessoriydintä ja laskentateho on 1.026 PFLOPS [22]. Suomen uudessa Louheksi nimitetyssä superkoneessa on 9432 prosessoriydintä ja 86.7 TFLOPS:in laskentateho.

Tietoverkkotekniikan kehittyessä ja halventuessa käsitys rinnakkaislaskennasta leveni edelleen. Tuli mahdolliseksi yhdistää joukko tehokkaita työasemia nopealla lähiverkolla klusteriksi, jota ohjelmoidaan viestinvälitysmallia tukevalla ohjelmistolla, kuten MPI [18]. Beowulf [4] on esimerkki “köyhän miehen” klusterista. Todella raskasta laskentaa tarvitsevat käyttävät useita supertietokoneita Gridverkon yli [13]. Kun tämä ajatustapa viedään äärimmilleen, voidaan kysyä, muodostavatko kaikki Internetiin kytketyt tietokoneet yhden suuren rinnakkaistietokoneen. SETI-projekti [20] edustaa tällaista rinnakkaislaskentaa. Top500-listalta [22] ilmenee, että teholaskennassa viimeisen kymmenen vuoden aikana jaetun muistin koneiden osuus on laskenut 50 prosentista olemattomiin, massiivisesti rinnakkaisten hajautetun muistin koneidenkin osuus on vähentynyt, kun klusterit ovat nousseet dominoivaan asemaan. Suurten prosessorimäärien löyhähkö yhteenkytkeminen on luonut raa’alta laskentateholtaan massiivisia konstellaatioita, mutta samalla kyseistä laskentatehoa hyödyntävien sovellusten määrä on supistunut. Pohjimmiltaan rinnakkaislaskennan marginalisoituminen johtuu siitä, että prosessorien välinen heikko tiedonsiirtokyky pakottaa karkeajakoiseen rinnakkaislaskentaan, joka ei voi tehokkasti levittyä suurelle määrälle laskentayksiköitä.

3 Rinnakkaislaskennan mallit

Tietokoneiden syntyhistoriassa on laskennan teoreetikoilla ollut merkittävä rooli. John von Neumann hahmotteli ohjelmoitavan koneen mallin, jossa muistiin tallennettu ohjelma määrittelee keskusyksikön toiminnan ja tietoa haetaan hajasaantimuistista keskusyksikköön, prosessoriin, käsiteltäväksi. Tästä syystä peräk-

käistietokonetta sanotaan von Neumannin koneeksi. Algoritmien teoriassa koneesta yleensä käytetään nimitystä RAM (Random Access Machine) eli hajasaantikone. Niin pian, kun alettiin rakentaa tietokoneita, joissa on monta prosessoria, tuli tarpeelliseksi laajentaa mallia. Von Neumann sai kyseenalaisen kunnian termissä "von Neumannin pullonkaula", joka tarkoittaa sitä, että yhden laskentayksikön ja muistin välinen siirtoväylä on erittäin rajoittunut muistin suureen määrään nähden. Kaiken laskennan suorittaminen yhdessä prosessorissa on laskentatehoa rajoittava tekijä. PRAM (Parallel RAM) [12] yleistää RAM-mallia niin, että samaa muistia käyttää joukko prosessoreita (kuva 1a). Mallissa oletetaan, että prosessorit voivat lukea ja kirjoittaa samanaikaisesti ja nämä operaatiot voidaan toteuttaa välittömästi. Rinnakkaisalgoritmien teoria nousi nopeasti kukoistukseen, ja parissakymmenessä vuodessa saatiin aika hyvä kuva, kuinka paljon rinnakkaisuutta laskennollisista tehtävistä löytyy [14]. Aika paljon löytyykin.

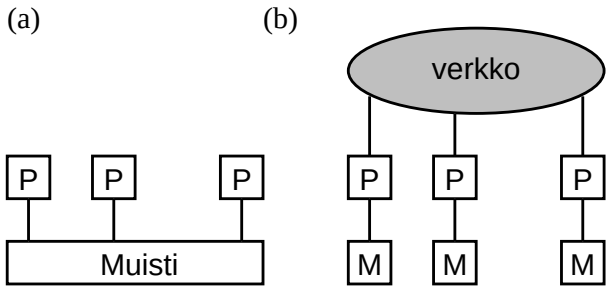
PRAM-malli on sikäli erinomainen laskennan malli, että RAM-malliin verrattuna ohjelmointikieleen riitti lisätä yksi ohjausrakenne, PARDO-silmukka, jonka sisällä olevat käskyt suoritetaan useassa prosessorissa samanaikaisesti. PRAM-mallin avulla voidaan kuvata, kuinka laskentatehtävä voidaan jakaa säikeiksi ja suorittaa suurella määrällä prosessoreja. Tästä syystä PRAM-algoritmikirjallisuus on rikas [14]. Oikeastaan juuri muuta kehittynyttä rinnakkaisalgoritmien teoriaa ei olekaan kuin PRAM-algoritmiikka.

Rinnakkaistietokoneiden suunnittelijat ja valmistajat käyttivät menneinä vuosikymmeninä PRAM-mallia jossain määrin osviittanaan. New York University Ultracomputer [2] ja SB-PRAM [3] ovat sellaisia koneita. Tera MTA (MultiThread Architecture) -rinnakkaistietokone [1] on sikäli merkittävä PRAM-arkkitehtuuri, että se on edelleen tuotannossa CRAY XMT -nimisenä.

Mikroprosessorin nopean kehityksen vuosina tietokonesuunnittelun valtavirta alkoi kuitenkin kehittyä toiseen suuntaan. Yhtäkkiä mikrotietokoneessa oli enemmän suoritustehoa kuin suuressa tietokonejärjestelmässä. Paljon tehoa vaativaan laskentaan oli edullisinta kytkeä yhteen joukko muistillisia prosessointiyksikköjä kuvan 1b tapaan. Viestinvälitysmallin mukaisesti ohjelmoija lähetti ohjelman osia toisiin käsittely-yksiköihin suoritettavaksi. Ideaalisen välittömän muistisaannin ja todellisen tietokoneen hitaan muistisaannin välillä alkoi olla liian suuri ero.

Vuonna 1993 julkaistua LogP-mallia [6] kutsuttiin "realistiseksi" rinnakkaislaskennan malliksi antaen ymmärtää, että PRAM on epärealistinen. Tässä mallissa kirjaimet LogP viittaavat koneen ominaisuuksiin: latenssi, kommunikoinnin aloituskustannus (overhead), kaistanleveydestä riippuva kommunikoinnin harvuus (gap) ja prosessorien määrä, jotka pitää ottaa huomioon ohjelmaa kirjoitettaessa. Malli on huomattavasti tarkempi kuin PRAM, mutta samalla se sälyttää ohjelmoijalle huomattavan vastuun tehokkuudesta. Samantapainen malli on BSP (Bulk Synchronous Parallel) [5]. Näiden mallien kohtalokas heikkous on, että kun ohjelmoija joutuu rinnakkaisalgoritmin suunnittelun ja viestinvälityksen lisäksi ottamaan huomioon nämäkin parametrit, ohjelmoinnista tulee vaikeaa.

Hiukan kärjistäen voidaan sanoa, että rinnakkaislaskennassa on ollut vain huonoja vaihtoehtoja: PRAM-mallin mukaista rinnakkaistietokonetta on helppo ohjelmoida mutta mahdoton rakentaa, kun taas LogP-malliin ja viestinvälitykseen sopiva kone on helppo rakentaa, mutta mahdoton



Kuva 1: Rinnakkaislaskennan mallit: (a) Jaetun muistin malli ja (b) hajautetun muistin malli.

ohjelmoida. On totta, että nykyisiä massiivisesti rinnakkaisia tietokoneita käytetään tehokkaasti tiettyihin tarkoituksiin, mutta asian kääntöpuoli on, että rinnakkaislaskennan käyttö on marginaalista verrattuna peräkkäiseen tietojenkäsittelyyn.

Naivi PRAM-mallin kritiikki kohdistuu oletukseen, että prosessorit saavat välittömästi datan muistista käsiteltäväkseen, mikä ei ole edes teknisesti mahdollista [8]. On kuitenkin olennaista ymmärtää, että vaikka todellisuudessa muistinkäsittelyyn liittyy latenssi, tämän hidasteen vaikutus laskennan etenemiseen voidaan tietyin keinoin häivyttää kokonaan. Tavoitetila on, että ohjelmoija voi olettaa tietokoneen muistinkäsittelyn toimivan kuin PRAM, *ilman latenssia*, vaikka fyysinen rinnakkaistietokone ei ole PRAM (samahan tapahtuu myös nykyisissä peräkkäistietokoneissa, jotka eivät fyysisesti noudata RAM-mallia, mutta ohjelmoija näkee ne sellaisina). Mikäli ohjelmassa on runsaasti toisistaan riippumattomia rinnakkain suoritettavia *säikeitä*, enemmän kuin prosessoreita, suoritusvuoroa odottavat säikeet ehtivät odotusaikana saamaan datansa muistista, ja prosessorit voivat koko ajan olla tehokkaassa käytössä. Tätä kutsutaan *rinnakkaisen pelivaran* (engl. slackness) periaatteeksi [23, 19]. Jotta kaikki säikeet saisivat odotusaikana tarvitsemansa datan, täytyy koneessa olla riittävästi *kaistanleveyttä* tietojen kuljettamiseen. Prosessorien määrän ja tiedonvälitysverkon täytyy olla tasapainossa. Toisaalta muistimoduulien pitää yhdessä kyetä käsittelemään aikayksikköä kohti keskimäärin yhtä paljon muistiviittauksia kuin prosessorit niitä yhteensä tuottavat. Vaikka muistimoduulit olisivatkin hyvin hitaita prosessoreihin verrattuna, riittävän nopea muistiviittausten käsittely on mahdollista, jos muistimoduuleja on paljon enemmän kuin prosessoreita ja muistiviittausten kasaantuminen yksittäisiin muistimoduuleihin onnistutaan välttämään.

Tiedonvälitysverkon ominaisuuksien lisäksi kriittinen vaatimus on toisistaan riippumattomien rinnakkaisten säikeiden suuri määrä. PRAM-malliin perustuva rinnakkaislaskennan teoria tarjoaa runsaasti algoritmeja, jotka ratkaisevat tehtävän nopeasti (“polylogaritmisessa” ajassa) käyttäen suurta määrää (virtuaalisia) prosessoreja, tekemällä likimain saman verran työtä kuin parhaat tunnetut peräkkäisalgoritmit. Koska peräkkäisalgoritmeissa ongelmien laskennallinen kompleksisuus on vähintään lineaarinen ongelman koon suhteen, nopeat PRAM-algoritmit käyttävät siis hyvin runsaasti rinnakkaisia säikeitä.

4 PRAM-mallin uusi lupaus

Rinnakkaislaskennan painopisteen siirtyminen 1990-luvulta alkaen viestinvälitysmalliin on toisaalta tuonut Beowulfin kaltaiset klusterit jokamiehen ulottuville ja mahdollistanut todella massiivisten prosessorimäärien käytön laskennassa, mutta samalla se on marginalisoinut rinnakkaislaskennan käytön harvoihin erikoistehtäviin. Vaikka ohjelmoinnissa voidaankin käyttää MPI:n kaltaista rajapintaa, ohjelmointi on arkkitehtuuririippuvaisena ja konekielimäisen matalatasoisena liian vaikeaa. Toisaalta ne oletukset, joiden perusteella PRAM-malli hylättiin 1990-luvulla, eivät enää täysin päde [11]. Keskittyminen peräkkäisprosessorin tehon nostoon on rajoittanut teknologisten mahdollisuuksien hyväksikäyttöä. Kummallinen todistus tästä on se, että henkilökohtaisessa tietokoneessa voi näytönohjaimena olla piiri, joka on monin verroin tehokkaampi rinnakkaistietokone kuin varsinainen prosessori. Nokkelimmat halvan laskentatehon etsijät ovatkin alkaneet “väärinkäyttää” grafiikkaprosessoria rinnakkaistietokoneena [7]. Entä jos näillä taidoilla olisi varta vasten yritetty rakentaa rinnakkaistietokone? Tätäkin on tutkittu.

4.1 PRAM-rinnakkaistietokoneet

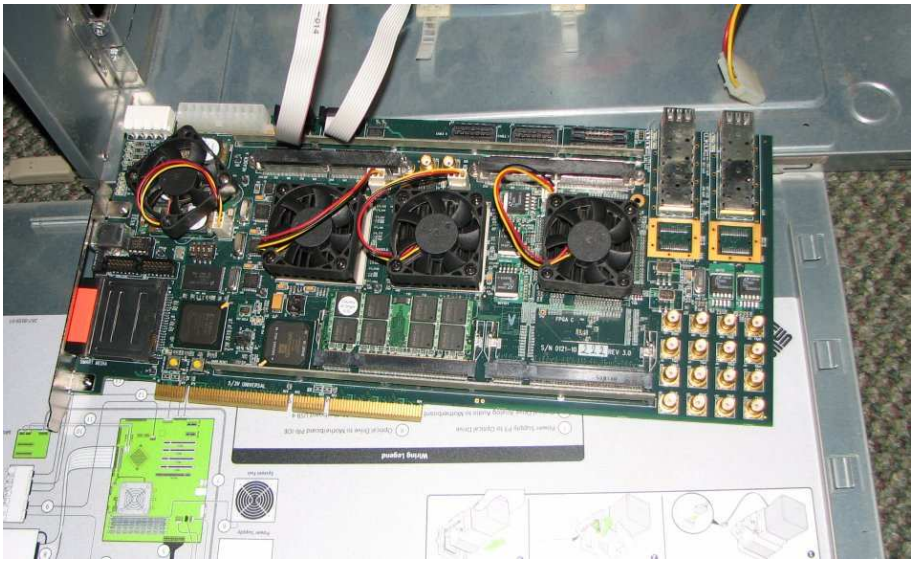
Saarbrückenin yliopiston SB-PRAM-projektissa 1990-luvun lopulla rakennettiin 64-prosessorinen modulirakenteinen SB-PRAM-niminen rinnakkaistietokoneen prototyyppi, joka tosin oli jo rakennusaikana tekniikaltaan vanhentunut. Sen ansioksi on kuitenkin luettava ohjelmointimallin kehittely ja monipuolinen c-kielen sukuinen rinnakkaisohjelmointikieli Fork [15]. Myös Teracomputerin (nyk. Cray) edelleen tuotannossa oleva MTA [1] kannattaa mainita PRAM-mallia toteuttavana rinnakkaistietokoneena.

Marylandin yliopiston XMT-projekti (eXplicit MultiThreading) on kehittänyt asynkroniseen PRAM-malliin perustuvan prosessorin ja sille c-kielen kaltaisen rinnakkaisohjelmointikielen ja toteuttanut sen FPGA-piireillä [24]. Tällainen PRAM on Chip -prosessori, jolle on sittemmin annettu nimi Paraleap, on todella olemassa ja siihen perustuvaa tietokonetta (katso kuvaa 2) voi Internetin kautta käyttää vaikka Suomesta.

Omassa tutkimussäikeessämme olemme tutkineet PRAM-mallin toteutettavuutta teoreettisten mallien [16], kommunikaatioarkkitehtuurien, prosessoriarkkitehtuurien ja kommunikaation kannalta. Eclipse-arkkitehtuurikehikon avulla on tutkittu eri PRAM-varianttien toteutusta moniytimellisellä mikropiirillä käyttäen syklitöntä moniristikkoverkkoa kommunikointiin [9, 10]. Eclipse-arkkitehtuurin pääideana on emuloida PRAM-laskentamallia prosessorimäärän suhteen skaalautuvalla hajautetun muistin tietokoneella siten, että globaalien muistiviittausten latenssi peitetään suorittamalla synkronisesti useita säikeitä prosessoriydintä kohti.

4.2 Kaksi prosessoriratkaisua

Nykyaikaisella prosessoripiirillä voi olla monta miljardia komponenttia. Prosessoriteollisuus on huomannut, että piiriltä voi saada enemmän laskentatehoa, kun sille rakennetaan yhden prosessorin asemesta useita prosessoriytimiä. Paraleap- ja Eclipse-prosessoreissa tämä ajattelu on viety astetta pitemmälle, sillä ne on alusta alkaen suunniteltu suorittamaan PRAM-tyyppistä hienojakoisesti rinnakkaista ohjelmaa.



Kuva 2: Paraleap tietokoneessa. Prosessori on vasemmalla kolmella FPGA-piirillä.

Paraleap-prosessorin rakenne on pelkistettynä kuvan 3 mukainen. MTCU (Master Thread Control Unit) on peräkäissuorituksesta ja koko prosessorin toiminnasta vastaava prosessori, P:t ovat rinnakkaissuorituksessa käytettäviä prosessointiyksiköitä (Paraleapissa 64), PSU on prefiksisummayksikkö ja Reg on joukko rekistereitä. Prosessorin sisäinen verkko yhdistää prosessorit ulkoiseen muistiin M välimuistien C kautta.

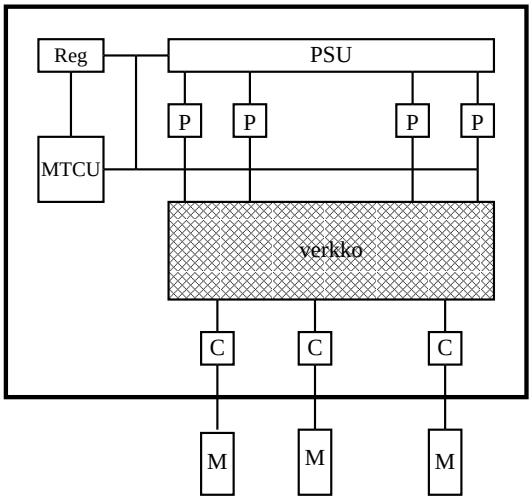
Prefiksisummayksiköllä on keskeinen rooli prosessorien työnjaossa, ja se voi palvella kaikkia prosessoreita käytännössä samanaikaisesti. Lukujonon $x_1, x_2, x_3, \ldots, x_n$ prefiksisumma on lukujono $x_1, x_1 + x_2, x_1 + x_2 + x_3, \ldots, x_1 + x_2 + \ldots + x_n$. Paraleap-prosessorin PSU toimii itse asiassa peräkkäisesti, mutta se on niin nopea, että operaation pyytäjät voivat kuvitella saavansa palvelun välittömästi. Seuraava esimerkki havainnollistaa, miten prefiksisumman avulla hallitaan rinnakkaisuutta. Vektorin kompaktointi eli nollaalkioiden poisto vektorista A voidaan suorittaa seuraavalla PRAM-ohjelmalla:

```
for i=0..n-1 pardo
  if A[i]=0
    then C[i]=0 else C[i]=1
E := prefix-sum(C)
for i=0..n-1 pardo
  if A[i]<>0
    then B[E[i]]:=A[i]
```

Prefiksisumma mahdollistaa varsinaisen kompaktoinnin (kaksi pardo-silmukkaa) suorittamisen ajassa $O(1)$.

Paraleap-prosessorin ohjelmointikielellä XMT-C sama ohjelma kirjoitetaan muodossa

```
psBaseReg x=0
spawn(0,n-1){
  int e;
  e=1;
  if (A[$]!=0) {
    ps(e,x);
    B[e]=A[$]; }}
```



Kuva 3: Paraleap-prosessorin rakenne.

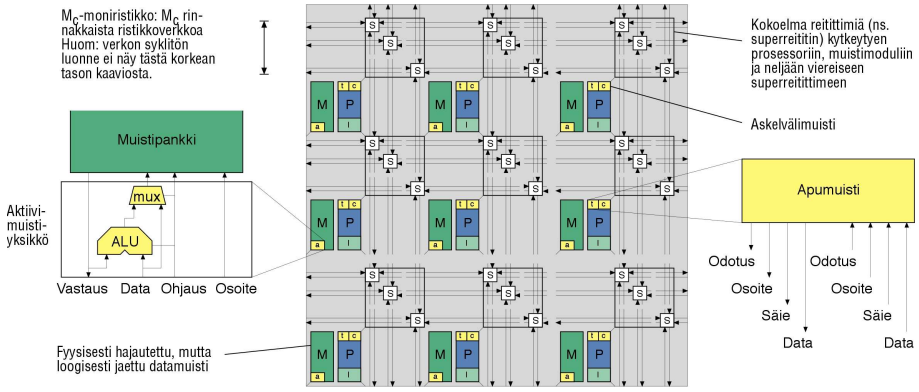
Tämä on Paraleap-tietokoneessa suoritettava ohjelma. Paraleap on todellakin fyysisesti olemassaoleva, toimiva prosessori. Prosessori on toteutettu 75 MHz:n FPGA-piirillä (Field Programmable Gate Array) kts. kuvaa 2. Seuraavaksi prosessori tullaan toteuttamaan 800 MHz:n ASIC-piirillä (Application-Specific Integrated Circuit), jolloin suoritus nopeutuu. Prototyypin testiajoista on arvioitu [24], että testiin kuuluneilla kahdeksalla ohjelmalla ASIC-piirillä toteutettava Paraleap olisi 1.6–9 kertaa nopeampi kuin 2.6 GHz:n AMD Opteron -prosessori.

Eclipse-prosessorin rakenne on kuvan 4 mukainen. Lisätehoa laskentaan haetaan yhdistämällä nykyisissäkin prosessoreissa käytetty käskytason rinnakkaisuus saumattomasti monisäiesuoritukseen ketjutustekniikan avulla, joka pystyy löytämään virtuaalista käskytason rinnakkaisuutta jopa peräkkäisestä ohjelmakoodista. Samaan muistipaikkaan kohdistuva yhtäaikainen viittaaminen on toteutettu ns. askelvälimuistien avulla. Niiden tarkoituksena on vähentää kunkin prosessorin tekemien viittausten määrää yhteen kutakin muistipaikkaa kohti. Keskeneräisten viittausten tietoja säilyttävää apumuistia ja operaatiot käytännössä laskevia aktiivimuistiyksikköjä taasen käytetään multiprefiksioperaatioiden toteuttamiseen. Eclipse-arkkitehtuurien käytännöllistä toteutettavuutta on arvioitu perusteellisella analyyttisella mallinnuksella [11] eikä esteitä tehokkaalle piitoteutukselle nykyteknologialla pitäisi enää olla. Prototyypin rakentaminen FPGA-piirille on suunnitteilla.

Eclipsen e-kielellä vektorin kompaktointi kirjoitetaan muodossa

```
if_( A_[_thread_id],
   prefix(e,MPADD,&x_,1);
   B_[e]=A_[_thread_id]; );
```

Paraleapin ja Eclipsen kaltaiset prosessorit ovat toivottavasti se “Kolumbuksen muna”, joka olemassaololollaan todistaa, ettei PRAM olekaan mahdoton vaan se on itseasiassa luonnollinen ja lupaava tapa saada prosessoripinta tehokkaaseen käyttöön nyt, kun peräkkäissuoritus-



Kuva 4: Eclipse-arkkitehtuuri (P = prosessori, M = datamuisti, I = käskymuisti, a = aktiivimuistiyksikkö, c = askelvälimuisti ja t = apumuisti).

sella ei enää saada lisätehoa irti. Mentaalisten esteiden murruttua prosessorisuunnittelijat vapautuvat kehittämään prosessoreita, jotka pystyvät tehokkaasti suorittamaan suurta määrää säikeitä, jotka PRAM-algoritmiikka tuottaa.

5 Rinnakkaiskoneen ohjelmointi

Prosessorien moniydinvarustelun myötä myös ohjelmointi muuttuu rinnakkaiseksi. Olemassa olevan ohjelmakoodin kannalta tämä ei ole välttämättä hyvä uutinen. Vanhoja peräkkäisohjelmia voidaan tietenkin suorittaa rinnakkain eri prosessoreilla kuten ennenkin, mutta ohjelman nopeampi suorittaminen edellyttää rinnakkaisalgoritmin käyttöä ja ohjelmointia rinnakkaisohjelmointikielellä. Joissakin tapauksissa peräkkäisohjelman automaattinen rinnakkaistuskin voi toimia.

Rinnakkaisessa ohjelmoinnissa PRAM-malli näyttänee suurimmat hyötynsä arkkitehtuuririippumattomana ja korkean abstraktiotason omaavana mallina: PRAM-tietokoneet voivat hyödyntää jaettujen muuttujien allokaatiota ilman vaatimusta käsiteltävän datan lokaalisuudesta tai pelkoa suorituskyvyn romahtamisesta. Ohjelmoija voi hyödyntää implisiittistä askeleittaista synkronointia erittäin hienorakeisten rinnakkaisalgoritmien toteuttamiseksi ilman huolta välimuistien koherenssiongelmista, eksplisiittisistä kommunikaatiokomennoista ja arkkitehtonisista parametreistä, jotka hankaloittavat merkittävästi muiden mallien ohjelmointia ja tekevät siitä arkkitehtuurisidonnaista.

Esimerkiksi viestinvälitysmalli MPI ei mahdollista hienorakeista rinnakkaislaskentaa, sillä jo MPI-kutsujen suoritus vie satoja tai tuhansia kellojaksoja ja pakottaa näin käsittelemään tietoa suuremmissa palasissa. MPI:n toinen ongelma liittyy siihen, että toiminnallisuuden mappaus ja datan partitiointi, synkronointi ja kommunikaatio ovat arkkitehtuurisidonnaisia ja täytyy tehdä MPI:ssä eksplisiittisesti ilman yhtenäisen algoritmiikan tukea. BSP- ja LogP-malleissa niinikään jo mallin olemus viittaa siihen, että hienorakeiset PRAM-algoritmit eivät ole käytettävissä. Algoritmien suunnittelun tekee hankalaksi se, että riippuen mallin parametreistä laskennallisen ongelman par-

haiten ratkaiseva algoritmi näyttää hyvin erilaiselta eikä kunnollista algoritmiikkaa ole näillekään.

Rinnakkaisuuden väistämättömyydestä seuraa, että ohjelmoinnin yliopisto- ja ammattikorkeakouluopetuksessa on syytä ottaa rinnakkaisuus teorioineen, malleineen ja työkaluineen heti alusta pitäen mukaan ohjelmoinnin käsitteistöön. Joka tapauksessa teollisuudessa tulee olemaan pula rinnakkaista ohjelmointia taitavista ohjelmoijista ja opettajista, ennen kuin nämä satsaukset alkavat tuottaa tulosta.

6 Kuinka eteenpäin?

Paraleapin ja Eclipsen kaltaiset prosessorit ovat mielestämme merkittävä päänavaus yritettäessä toteuttaa rinnakkaisuutta piiritasolla. Prosessorien työnjakoa ei välttämättä kannata toteuttaa juuri sellaisella PS-yksiköllä kuin Paraleapissa on, eikä prosessorin sisäisen verkon tarvitse olla sellainen puuristikko kuin Paraleapissa. Nämä prosessoriarkkitehtuurit ovat kuitenkin osoittaneet toimintakykynsä ja tuovat uusia raikkaita ajatuksia prosessoriteollisuuteen.

Paraleap ja Eclipse ovat ratkaisuehdotuksia, miten toteuttaa rinnakkaisuus mikropiiritasolla. Suuressa laskennassa yksi prosessori ei riitä. Miten suuri rinnakkaistietokone tulisi rakentaa tekemättä ohjelmoijan työtä vaikeaksi? PRAM-prosessoreista pitäisi voida rakentaa suuri PRAM-tietokone niin, että ohjelmoija voi edelleen ajatella samanlaista PRAM-mallia, jossa rinnakkaisuus ilmaistaan PARDO-rakenteella. Fyysinen ongelma on rakentaa suuri PRAM pienemmistä PRAM-moduuleista. Ohjelmien allokoinnissa voitaneen käyttää hyväksi sitä, että rinnakkaisalgoritmien suunnittelussakin lohkominen on yleisesti käytetty tekniikka. Tällöin ohjelmalohkot sijoittuisivat luonnollisella tavalla PRAM-moduuleihin

Tietojenkäsittelyssä on ohjelmoija edelleen avainroolissa. Peräkkäistietokoneiden aikakaudella koulutetut ohjelmoijat on nimenomaisesti opetettu sarjallistamaan kaikki suorittaminen yhdessä prosessorissa suoritettavaksi. Rinnakkaistietokoneiden aikakaudella ohjelmoijan pitää oppia ajattelemaan, mitä kaikkea voi suorittaa rinnakkain. Rinnakkaisohjelmointi ei ole välttämättä vaikeampaa tai helpompaa kuin peräkkäisohjelmointi, mutta se vaatii erilaista ajattelua ja uutta linjaa ohjelmoinnin opetuksessa.

Viitteet

[1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, B. Smith. “The Tera computer system.” In *Proc. of the 4th International Conference on Supercomputing*, 1990.

[2] G.S. Almasi, A. Gottlieb. *Highly Parallel Computing.* Benjamin/Cummings, 1994.

[3] P. Bach, M. Braun, A. Formella, J. Friedrich, T. Grun, C. Lichtenau. “Building the 4 processor SB-PRAM prototype.” In *Proc. of the 30th Hawaii International Conference on System Sciences: Advanced Technology Track – Vol. 5*, 1997.

[4] `http://www.beowulf.org`

[5] `http://www.bsp-worldwide.org`

[6] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken. “LogP: Towards a realistic model of parallel computation.” In

Principles Practice of Parallel Programming, pp. 1–12, 1993.

[7] `http://www.nvidia.com/cuda`

[8] M. Forsell. “Are Multiport Memories Physically Feasible?” *Computer Architecture News 22(4)* (September 1994), pp. 47–54.

[9] M. Forsell. “A Scalable High-Performance Computing Solution for Network on Chips.” *IEEE Micro 22(5)* (September-October 2002), pp. 46–55.

[10] M. Forsell. “Realizing Multioperations for Step Cached MP-SOCs.” In *Proc. SOC’06*, November 14–16, 2006, Tampere, pp. 77–82.

[11] M. Forsell, J. Roivainen. “Performance, Area and Power Trade-Offs in Mesh-based Emulated Shared Memory CMP Architectures.” In *Proc. PDPTA’08*, Jul 14–17, 2008.

[12] S. Fortune, J. Wyllie. “Parallelism in Random Access Machines”. In *Proc. 10th ACM Symposium on Theory of Computing*, pp. 114–118, 1978.

[13] `http://www.grid.org`

[14] J. Jájá. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.

[15] J. Keller, C. Kessler, and J. Träff. *Practical PRAM Programming*. Wiley, 2001.

[16] V. Leppänen. *Studies on the realization of PRAM*. Dissertation 3, Turku Centre for Computer Science, University of Turku, 1996.

[17] G.E. Moore. “Cramming more components onto integrated circuits”. *Electronics* 38(8), 1965.

[18] `http://www.mpi-forum.org`

[19] M. Penttonen. “Rinnakkaisalgoritmit ja rinnakkaistietokoneet”. *Tietojenkäsittelytiede* 26:9–21.

[20] `http://setiathome.ssl.berkeley.edu`

[21] A. Seznec. “5mm x 15 mm: the new frontier of parallel computing.” *Euro-Par 2007 keynote*, August 28–31, 2007, Rennes, France.

[22] `http://www.top500.org/list/2008/06/100`, kesäkuu 2008.

[23] L.G. Valiant. “General Purpose Parallel Architectures”. In *Handbook of Theoretical Computer Science, Ed. Jan van Leeuwen, Elsevier and MIT Press*, volume 1. Elsevier Science, 1990.

[24] X. Wen, U. Vishkin. “FPGA-based prototype of a PRAM-On-Chip processor.” *Computer Frontiers 2008*, May 5–7, 2008.