

S-114.240 Rinnakkaislaskenta laskennallisessa tieteessä:

## MPI ja työasemaklusterit: Beowulf

Risto Virkkala

Laskennallisen tekniikan laboratorio

5.5.1999

Muokattu:

5.7.1999 (testit 16 koneen Beowulf klusterilla)

17.8.1999 (Antti Kurosen testitulokset 16 koneen klusterilla moldy simulaatiolla)



# 1. Johdanto

Perinteisesti supertietokoneet ovat olleet suurten tietokonevalmistajien toimittamia koneita, jotka sisältävät juuri supertietokonekäyttöön suunniteltua laitteistoa. Suunnittelun ensimmäisenä lähtökohtana on ollut maksimaalinen teho ja sen seurauksena laitteet ovat olleet hyvin kalliita. Henkilökohtaisten työasemien hintojen laskiessa ja suorituskyvyn jatkuvasti noustessa on kuitenkin normaalien työasemien yhdistäminen nopealla verkolla klusteriksi muodostunut yhä houkuttelevammaksi vaihtoehdoksi. Työasemaklusteri (Cluster of Workstations, COW) sisältää yleensä noin 4-140 normaalia työasemaa, jotka on omistettu vain rinnakkaisohjelmien suorittamiseen. Kyseisillä työasemaklustereilla on mahdollista päästä hyvin suureen hinta/suorituskyky suhteeseen. Klustereissa käytetään nykyään yleisesti myös PC-tason tietokoneita (Pile of PCs, PoPC) niiden suorituskyvyn lähestyttyä perinteisten työasemien suorituskykyä. Työasemaklusterina voidaan käyttää myös normaalissa työkäytössä olevaa työasemaverkkoa (Network of Workstations, NOW), jolloin verkossa olevien työasemien joutokäyntiaika saadaan hyödynnettyä rinnakkaisohjelmien suorittamiseen. Työasemaverkon käytössä on kuitenkin omat kuorman tasaukseen ja verkon suorituskykyyn liittyvät ongelmansa.

Työasemaklusterin suorituskykyä ei suoraan pystytä hyödyntämään perinteisesti kirjoitetulla yhdessä tietokoneessa suoritettavaksi tarkoitetulla ohjelmalla, vaan sovellus on varta vasten kirjoitettava rinnakkaistuvaksi. Yleisimmät rinnakkaisohjelmien kirjoittamiseen tarjolla olevat kirjastot ovat MPI (Message Passing Interface) ja PVM (Parallel Virtual Machine). Näistä MPI on saanut standardin aseman ja tässä selostuksessa käsitellään lähinnä sitä.

## 2. Beowulf työasemaklusterit

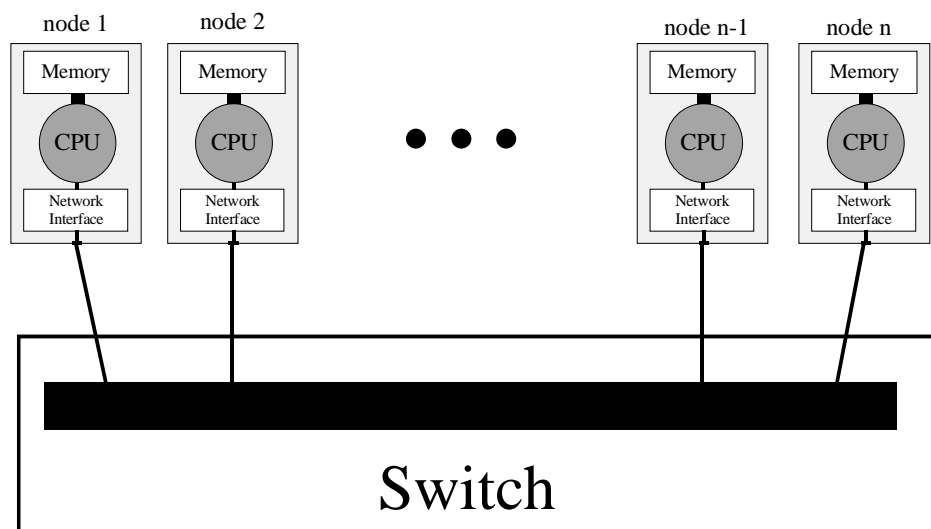
MPP-koneet (Massively parallel processor) ovat järjestelmiä, jotka on tehty kaikkein suurinta laskutehoa vaativiin sovelluksiin. Ne koostuvat normaaleista prosessoreista, jotka on liitetty yhteen supernopealla verkkoratkaisulla. MPP-koneet skaalautuvat satoihin jopa tuhansiin prosessoreihin [1]. Niiden haittana on kuitenkin kallis ja suurta suunnittelutyötä vaativa verkkoratkaisu sekä käyttöjärjestelmä. Vuonna 1994 NASAn Goddard Space Flight Centerissä aloitettiin projekti jonka tavoitteena oli tuottaa gigaflops nopeuteen pystyvä työasemaklusteri alle 50000 dollarin hintaan. Jotta hintatavoitteessa pysyttiin, täytyi prototyypisysteemissä käyttää vain vakiokomponentteja ja ilmaista open-source ohjelmistoa. Käyttöjärjestelmäksi valittiin Linux. Prototyypikone nimettiin Beowulfiksi. Kyseinen kone ei kuitenkaan saavuttanut kuin 60 Mflops-nopeuden [2]. Vuonna 1996 projekti pääsi tavoitteeseensa ja sai rakennettua 1.25 Gflops nopeudella toimivan koneen 50000 dollarin hintaan käyttämällä Pentium Pro prosessoreja ja 100Mbit/s nopeuksista fast ethernet verkkoa [3]. Tämän jälkeen useat yliopistot ja muut tutkimuslaitokset ovat rakentaneet omia Beowulf-klustereitaan. Nopeimpana vakiotekniikalla toteutettuna Beowulf-klusterina mainittakoon maailman 113. nopein tietokone Avalon 48.6 Gflops nopeudella, joka koostuu 140 Alpha-työasemasta (tieto lokakuulta 1998) [4]. Beowulf-klustereita käytetään monenlaisten eri tieteellisten ongelmien ratkaisuun esim. molekyyliidynamiikka, osittaisdifferentiaaliyhtälöt, MC-simulaatiot [5], sähkömagnetiikan simulaatiot [6] sekä suurten yhtälöryhmien ratkaisu.

Itse Beowulf nimellä tarkoitetaan yleisesti ottaen rinnakkaislaskentaan omistettua Linux työasemaklusteria rinnakkaisohjelmointikirjastoineen.

### ***Arkkitehtuuri***

Työasemaklusterit ovat hajautetun muistin koneita (Kuva 1). Jokainen prosessori suorittaa omaa kopiotaan ohjelmasta, joten kyseessä on MIMD (Multiple Instruction

Multiple Data) tyyppinen rinnakkaiskone. Kaikki eri prosessorien välinen kommunikaatio tapahtuu siis tietoliikenneverkon kautta. Tämä asettaa vaatimuksia nimenomaan käytettävälle verkkototeutukselle sekä rinnakkaisohjelmien prosessorien välisen kommunikaation määrälle. Työasemaklusterin suorituskyvyn pullonkaulana onkin juuri käytetty verkkoratkaisu.



Kuva 1: Työasemaklusterien arkkitehtuuri

Yleisin käytetty verkkoratkaisu on kytketty 100 Mbit/s fast ethernet. Kyseisellä verkkoratkaisulla saavutetaan noin 100  $\mu$ s latenssi tiedonsiirrossa ja siirtonopeus noin 10 megatavua sekunnissa kahden prosessorin välisessä tiedonsiirrossa. Vertailun vuoksi CRAY T3E:n latenssi tiedonsiirrossa on 1  $\mu$ s ja tiedonsiirtonopeus 600 megatavua sekunnissa [7]. Beowulf verkkoratkaisun nopeuttamiseksi voi yhteen tietokoneeseen liittää kaksi verkkokorttia, jolloin tiedonsiirtonopeudeksi saadaan noin 20 megatavua sekunnissa. Tiedonsiirron latenssia tämä ei kuitenkaan nopeuta. Ethernetin latenssia voidaan yrittää pienentää paremmilla ohjelmistoratkaisuilla. Siirtämällä verkkoajurit osittain Linuxin ytimeistä käyttäjätilaan ja käyttämällä kevyempää protokollaa kuin TCP/IP voidaan ethernetin latenssia pienentää merkittävästi [19].

Työasemaklustereihin on tarjolla myös muita verkkoratkaisuja, joista monet ovat nopeampia kuin fast ethernet. Esimerkiksi myrinet (latenssi 4.8  $\mu$ s, siirtonopeus noin 100 megatavua sekunnissa) [8]. Näiden erikoisverkkoratkaisujen ongelmana on hinta. Pelkkä verkko saattaa tulla maksamaan yhtä paljon kuin kaikki muut osat yhteensä.

Seuraavan sukupolven version ethernetistä (Gigabit ethernet) odotetaan tarjoavan yleistyessään kohtuuhintaisen ratkaisun lisätä verkon suorituskykyä. Sen teoreettinen siirtonopeus on kymmenkertainen fast ethernetiin verrattuna.

Suurin osa Beowulf-klustereista on koottu joko Intel- tai Alpha-työasemista. Esimerkiksi Laskennallisen tekniikan laboratorion Beowulf-klusteri koostuu 16 kappaleesta 600 MHz Alpha Linux-työasemaa. Yleensä klustereissa on yksi tietokone varattuna interaktiiviseen työskentelyyn, ohjelmien kääntämiseen ja käynnistämiseen ja loput on omistettu pelkästään laskentaan. Myös laskentakoneiden välillä oleva tiedonsiirtoverkko on yleensä eristetty ulkopuolisesta verkosta ja ulkopuolinen verkkoyhteys on yhdistetty vain interaktiiviseen työskentelyyn tarkoitettuun koneeseen.

Jos klusteri koostuu heterogeenisestä laiteympäristöstä, on kuorman tasaukseen kiinnitettävä erityistä huolta ohjelmoissa. Tällöin laskenta on jaettava eri prosessoreille niiden tehojen suhteessa [9]. Tämä vaikeuttaa rinnakkaisohjelmien suunnittelua huomattavasti.

## **Ohjelmisto**

Beowulf-klustereiden käyttöjärjestelmänä on Linux. Klusteriin on asennettu tarvittavat kirjastot rinnakkaisohjelmien kääntämistä ja suorittamista varten. Lisäksi on ohjelmia klusterin ylläpitoon ja valvontaan. Yleisin ja standardin aseman saavuttanut rinnakkaisohjelmien kirjoittamiseen tarkoitettu kirjasto on MPI [10]. Se on käännetty monelle eri tietokonearkkitehtuurille, joten sillä toteutetut ohjelmat ovat helposti siirrettävissä eri ympäristöihin. Yleisimmät kääntäjät ovat c, Fortran 77 ja Fortran 90.

### 3. MPI

MPI-viestinvälityskirjastolla tehdyssä ohjelmassa joukko prosessoreja suorittaa normaalia peräkkäisohjelmaa, jossa tiedonsiirto eri prosessorien välillä on toteutettu kutsuilla MPI-kirjastoon [11] [12]. Yleensä kaikki prosessorit suorittavat samaa ohjelmaa (SPMD, Single Program Multiple Data), mutta eri prosessorien on mahdollista myös suorittaa eri ohjelmaa (MPMD, Multiple Program Multiple Data). Tässä luvussa tutustutaan MPI:llä tehtyjen ohjelmien kääntämiseen ja suorittamiseen Beowulf-klusterissa. MPI toteutuksena on MPICH [13].

MPICH-kirjasto tarjoaa ohjelmien kääntämiseen omat komenkonsa. Komennot on lueteltu taulukossa 1.

Kieli	kääntäjä
Fortran 77	<code>mpif77</code>
Fortran 90	<code>mpif90</code>
c	<code>mpicc</code>
c++	<code>mpicc</code>

Taulukko 1

Kyseiset komennot kutsuvat MPICH-kirjaston asennuksen yhteydessä määriteltyjä järjestelmän omia kääntäjiä tarvittavin parametrein, joilla MPI-kirjastot linkitetään mukaan. Ohjelmat voi myös kääntää kutsumalla suoraan järjestelmän omia kääntäjiä tarvittavin parametrein. Käytettäessä MPICH:n omia käskyjä etuna on se, että käyttäjän ei varsinaisesti tarvitse tietää missä MPI-kirjastot sijaitsevat.

Tarkastellaan esimerkkinä yksinkertaista ohjelmaa, joka laskee piin likiarvon numeerisesti integroimalla:

$$\int_0^1 \frac{4}{1+x^2} dx = 4 \int_0^1 \arctan(x) dx = \pi$$

$$\Rightarrow \sum_{i=1}^N \frac{4}{1+x_i^2} \frac{1}{N} \xrightarrow{N \rightarrow \infty} \pi, \quad x_i = \frac{1}{N}(i-0.5) \quad (1)$$

Ohjelma (fpi.f, kuva 2) on toteutettu siten, että jokainen prosessori laskee oman osasummansa numeerisesta integraalista (1). Lopuksi eri prosessorien laskemat osasummat summataan yhteen MPI\_REDUCE komennolla.

```

program main
include 'mpif.h'

double precision mypi, pi, h, sum, x, f, a
integer n, myid, numprocs, i, rc

c                               function to integrate
f(a) = 4.d0 / (1.d0 + a*a)

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
print *, "Process ", myid, " of ", numprocs, " is alive"

10  if ( myid .eq. 0 ) then
    write(6,98)
98   format('Enter the number of intervals: (0 quits) ')
    read(5,99) n
99   format(i10)
endif

c                               broadcast problem size to all processes
call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

if ( n .le. 0 ) goto 30

h = 1.0d0/n
sum = 0.0d0
do 20 i = myid+1, n, numprocs
    x = h * (dble(i) - 0.5d0)
    sum = sum + f(x)
20  continue
mypi = h * sum

c                               collect all the partial sums
call MPI_REDUCE(mypi,pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
$ MPI_COMM_WORLD,ierr)

c                               node 0 prints the answer.
if (myid .eq. 0) then
    write(6, 97) pi
97   format(' pi is approximately: ', F18.16)
endif

goto 10

30  call MPI_FINALIZE(rc)
stop
end

```

Kuva 2: fpi.f piin likiarvon laskennan MPI-koodi

Esimerkkiohjelman kääntäminen tapahtuu `mpif77` komennolla:

```
gorgon ~/fpi 5 % mpif77 fpi.f -o fpi
```

Koska rinnakkaisohjelma on erikseen käynnistettävä jokaisessa klusterin prosessorissa, on käytettävä erillistä käynnistysohjelmaa `mpirun`. Esimerkkiohjelma käynnistetään neljälle prosessorille seuraavasti:

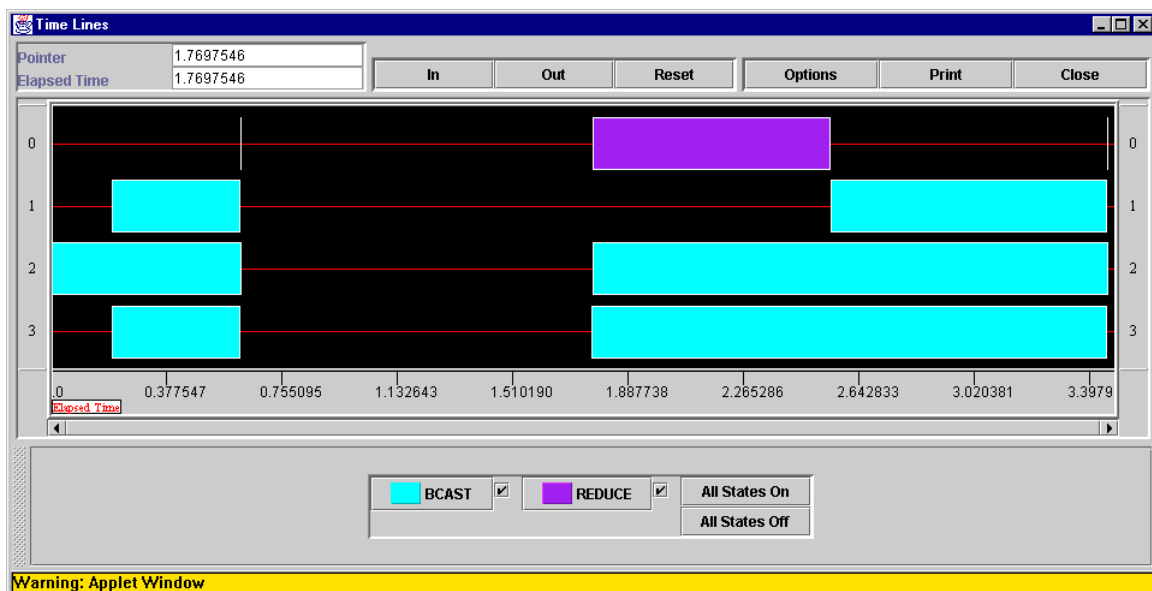
```
gorgon ~/fpi 6 % mpirun -nolocal -np 4 fpi
Process 1 of 4 is alive
Process 2 of 4 is alive
Process 3 of 4 is alive
Process 0 of 4 is alive
Enter the number of intervals: (0 quits)
100000000
pi is approximately: 3.1415926535902168
Enter the number of intervals: (0 quits)
0
```

Parametrilla `-np x` määrätään käytettävien prosessorien lukumäärä ja `-nolocal` tarkoittaa, että paikallisella koneella ei käynnistetä prosessia.

MPICH tarjoaa työkalut ohjelman suorituksen etenemisen seurantaan. Kääntämällä profiointikirjastot ohjelmaan mukaan jokaisesta MPI-kirjaston kutsusta ja sen kestosta tehdään lokimerkintä. Suorituksen jälkeen `loki` on mahdollista tarkistella `jumpshot` ja `upshot` -ohjelmilla. Kyseisten ohjelmien avulla voi tarkastella prosessien välisen kommunikaation viemää aikaa suhteessa laskentaan ja havaita mahdollisia epätasaisuuksia eri prosessorien kuormituksessa. Profiointikirjastojen kanssa esimerkkiohjelma käännetään seuraavasti:

```
gorgon ~/fpi 9 % mpif77 fpi.f -o fpi -lfmpich -llmpi -lmpe -lpmpich
```

Ohjelma suoritetaan `mpirun`:lla samoin kuin edellä. Suorituksen päättyessä levyllä on lokitiedosto nimeltä `fpi.clog`. `jumpshot` on java-ohjelma, jolla `clog`-tiedostoja voidaan tarkastella. Kuvassa 3 on esimerkki `jumpshot`:n ikkunasta kun tarkasteltavana on esimerkkiohjelman ajosta muodostunut loki. Jokainen rivi kuvassa kuvaa yhtä prosessoria. Vaakaviiva tarkoittaa aikaa, jonka ohjelma on käyttänyt laskentaan. Palkki viivan päällä kuvaa MPI-kirjastossa kulunutta aikaa. Kuvasta nähdään, että prosessorit 1-3 suorittavat `MPI_BCAST` komennon heti käynnistyksessä kun taas prosessori 0 odottaa ensin käyttäjän syöttämää ongelman kokoa ja suorittaa `MPI_BCAST` komennon vasta sen jälkeen noin hetkellä  $t=0.7$  s. Prosessorit 0, 2 ja 3 saavat oman laskentansa valmiiksi eli suorittavat `MPI_REDUCE` komennon noin hetkellä  $t=1.8$  s. Prosessori 1 oli testitarkoitusta varten tässä testiajossa noin puolet hitaampi ja se pääsee suorittamaan `MPI_REDUCE` komennon vasta, kun  $t=2.5$  s. Prosessorit 0,2 ja 3 ovat siis tyhjäkäynnillä ajan  $t=1.8$ s – 2.5 s. Tämän jälkeen prosessori 0 odottaa taas käyttäjältä uutta ongelman kokoa ja se siirretään kaikille prosessoreille taas `MPI_BCAST` komennolla, jonka jälkeen suoritus lopetetaan, koska uusi annettu ongelman koko on 0.



Kuva 3: `jumpshot`in ruutu.

Erityisesti monimutkaisemmilla MPI-ohjelmilla `jumpshot`ista ja `upshot`ista on hyötyä tarkasteltaessa kommunikaation viemää aikaa ja eri prosessorien kuormitusta.

MPI:n seuraava versio MPI-2 on jo kehitteillä. MPI-2 uudistuksia [14] ovat mm. dynaaminen prosessien hallinta ja rinnakkaistettu kommunikaatio.

## 4. Muita rinnakkaislaskentaan soveltuvia kirjastoja

MPI tarjoaa käyttäjälle vain perustoiminnot prosessorien väliseen kommunikaatioon. Esimerkiksi matriisinkäsittelyoperaatioita ei ole valmiina, vaan ne pitää itse toteuttaa MPI:n päälle. Rinnakkaislaskentaan optimoitujen matriisirutiinien kirjoittaminen on erittäin vaativa ja aikaa vievä projekti. Onneksi MPI:n päälle on valmiiksi toteutettuja kirjastoja, jotka tarjoavat rinnakkaistetut matriisinkäsittely operaatiot.

### **ScaLAPACK**

ScaLAPACK-kirjastossa [15] on rinnakkaistetut versiot osasta LAPACK:n tiheiden ja nauhamatriisien käsittelyrutiineita. ScaLAPACK rakentuu PBLAS-kirjaston päälle, jossa on rinnakkaistettu normaalin BLAS-kirjaston rutiineita.

Muita ScaLAPACK projektiin kuuluvia kirjastoja ovat:

PARPACK	Harvojen matriisien ominaisarvojen ja $-$ vektorien etsintään. (ARPACK:n rinnakkaisversio)
MFACT	Harvan lineaarisen yhtälöryhmän ratkaisuun
ParPre	Esikäsittelijä (preconditioner) iteratiivisille lin.yht. ratkaisimille

ScaLAPACK on tehty Fortran 77:lla, mutta sitä voi käyttää myös c-ohjelmista.

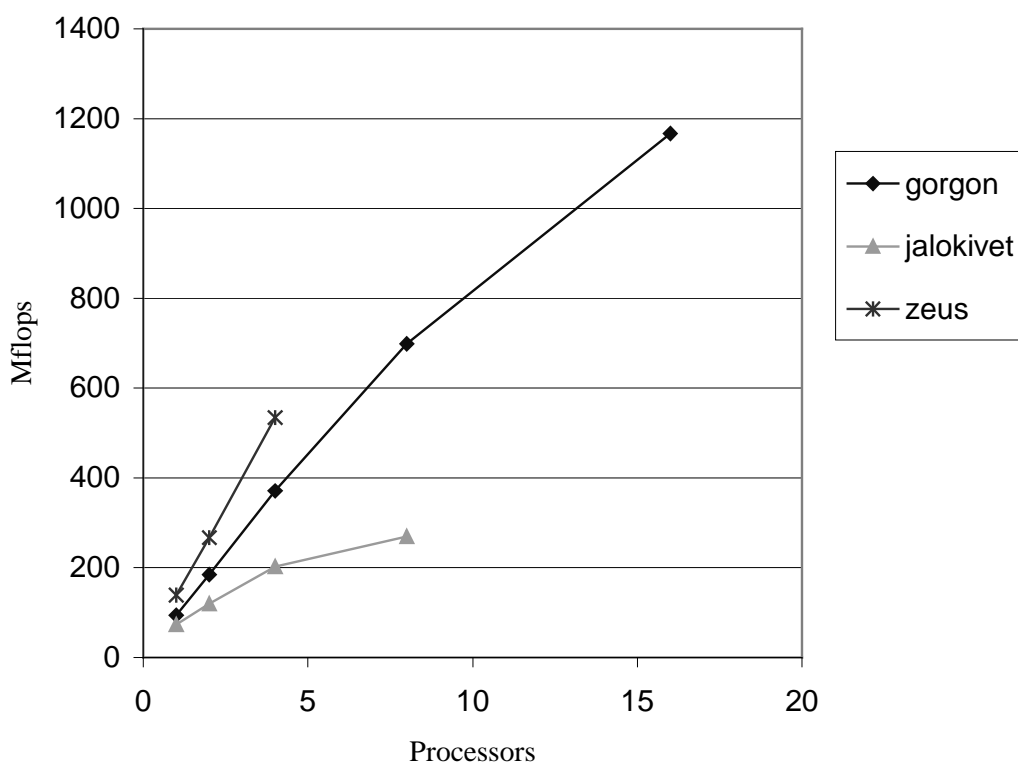
### **PETSc**

PETSc-kirjasto [16] tarjoaa yleisiä työkaluja tieteellisten ongelmien rinnakkaistamiseksi. Paketti sisältää esimerkiksi rutiinit harvojen matriisien käsittelyyn ja harvan lineaarisen yhtälöryhmän ratkaisemiseen. PETSc on tehty c:llä, mutta sitä voi käyttää myös Fortran ohjelmista.

## 5. Beowulf-klusterien skaalautuvuus

Rinnakkaisohjelmien skaalautuvuus on ongelmakohtainen. Mitä vähemmän prosessorien välistä kommunikaatiota on, sitä paremmin ongelma skaalautuu. Raportteja eri skaalautuvuuksista löytyy Internetistä laidasta laitaan. Avalonin kerrotaan saavuttavan 80%-90% rinnakkaistumishyötysuhde 70 prosessorilla suurella molekyyliidynamiikkasimulaatiolla [17]. FDTD simulaatio on rinnakkaistunut kohtalaisesti [6]. Toisaalta esimerkkinä huonosta skaalautumisesta on virtausdynamiikan ohjelma, jossa 16 prosessorilla 80% suoritusajasta kuluu kommunikaatioon [6]

Kuvassa 4 on NAS:n NPB 2.3 LU-testiohjelmalla [18] laskettuja tuloksia eri rinnakkaiskoneilla. Kyseinen ohjelma ratkaisee Navier-Stokes yhtälöt 64x64x64 hilassa. Testissä käytetyt koneet on lueteltu taulukossa 2.



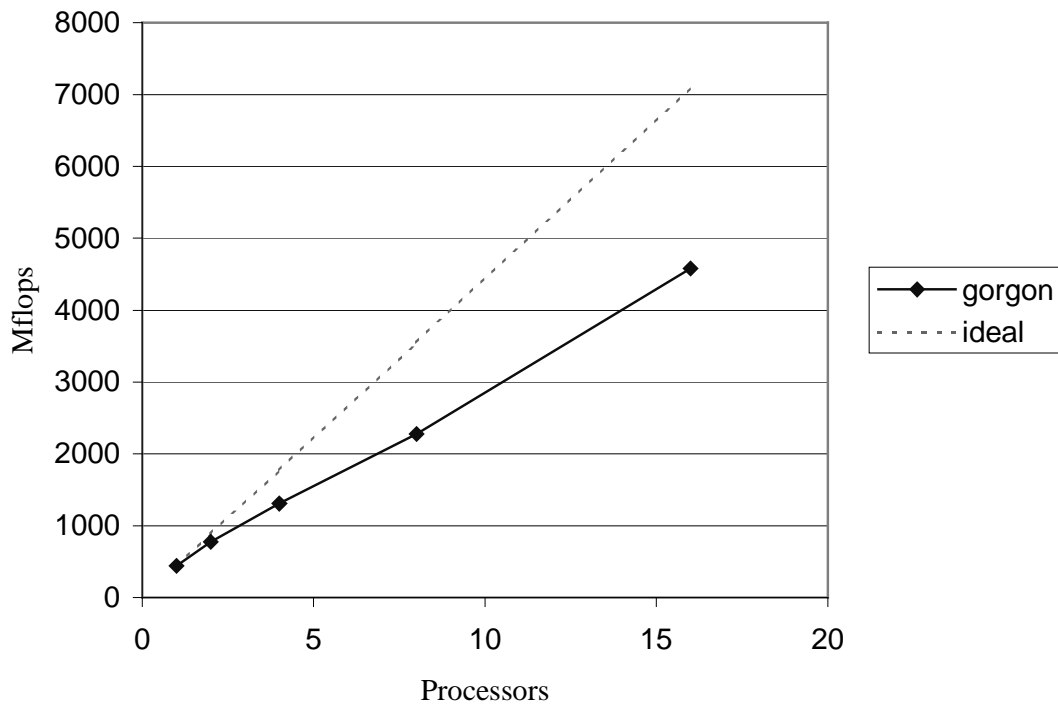
Kuva 4: NAS LU-testin tulokset. Liukulukuoperaatioiden määrä sekunnissa.

gorgon	16 x alpha 600 MHz Beowulf-työasemaklusteri (Linux) (Laskennallisen tekniikan laboratorio)
jalokivet	8 x alpha 433 MHz työasemaverkko (Digital Unix) (TKK:n laskentakeskuksen Maarintalon työasemaluokka)
zeus	4 x alpha 600 MHz jaetun muistin moniprosessori kone (Digital Unix) (Laskennallisen tekniikan laboratorio)

Taulukko 2

Testiä varten kaikki ohjelmat käännettiin samoilla kääntäjillä ja samoilla optioilla. Laskentakeskuksen jalokivet-koneissa oli samaan aikaan myös muuta kuormitusta, jalokivet-koneita voidaankin pitää työasemaverkkona eikä varsinaisena työasemaklusterina. Zeuksen Gorgonia parempaa tulosta yhdellä prosessorilla selittää osaltaan Zeuksen suurempi prosessorin välimuisti. Myös käyttöjärjestelmällä saattaa olla pientä vaikutusta. Gorgon Beowulf-klusteri skaalautuu tällä testillä neljään prosessoriin asti lähes ideaalisesti. Kahdeksalla prosessorilla rinnakkaistumistehokkuus on 93% ja kuudellatoista. 78%.

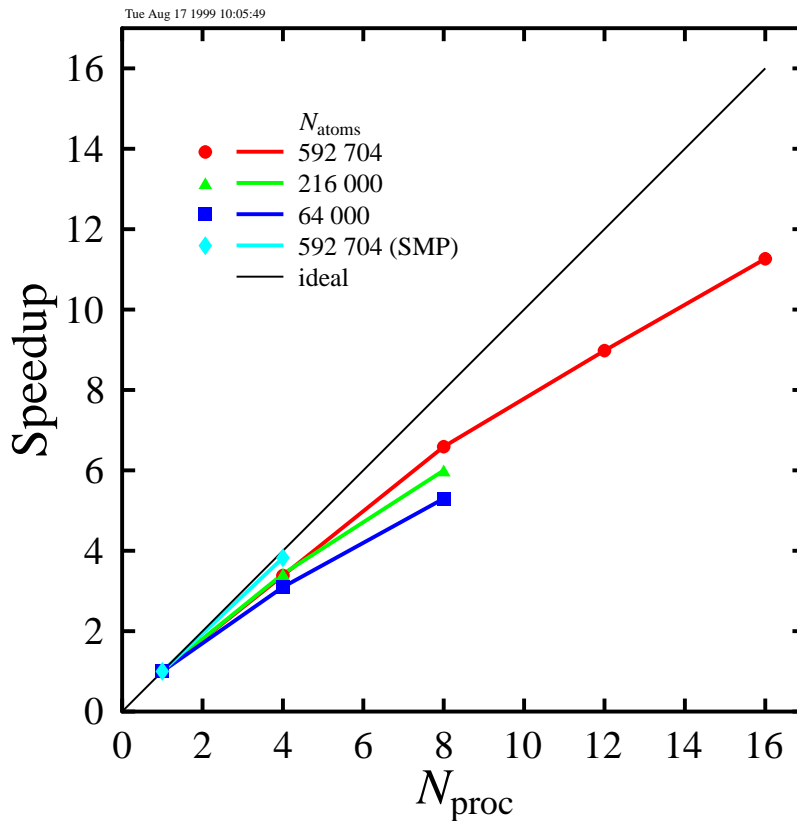
Toisena testinä Gorgonissa ajettiin ScaLAPACK-kirjastolla tiheän matriisin LU-hajotelma ja yhtälöryhmän ratkaisu. Matriisin koko on prosessorimäärä kahdennettaessa kerrottu noin  $\sqrt{2}$ :lla, jolloin jokaisella prosessorilla olevan datan määrä pysyy suunnilleen vakiona koko testin ajan. Matriisin koot testissä olivat: 3500x3500 ( $Np=1$ ), 5000x5000 ( $Np=2$ ), 7000x7000 ( $Np=4$ ), 10000x10000 ( $Np=8$ ) ja 14000x14000 ( $Np=16$ ). Tulokset ovat kuvassa 5.



Kuva 5: Liukulukuoperaatioiden määrä sekunnissa. LU-hajotelma ja yhtälön ratkaisu.

Rinnakkaistumistehokkuus on 16 prosessorin tapauksessa noin 64%. Tiheän matriisin tapauksessa prosessorien välistä tietoliikennettä on enemmän, joka näkyy heti suorituskyvyssä verrattuna edelliseen testiin. ScaLAPACK kirjaston parametrien valinnalla oli myös suuri vaikutus suorituskykyyn. Huonolla valinnalla saatettiin jäädä alle puoleen nyt saavutetusta tehosta.

Kolmantena suorituskykytestinä on Antti Kurosen laskemat molekyyliidynamiikka-simulaatioajon kestot (kuva 6). Ongelman koon kasvaessa tehokkuus lisääntyy, koska kommunikaation määrä suhteutettuna laskentaan pienenee. Isoimmalla ongelman koolla (592 704 hiukkasta) ja 16 prosessorilla päästään noin 72% rinnakkaistumistehokkuuteen. SMP testi on tehty Sun Ultra 4 Enterprise Server 450 tietokoneella, jossa on 4 kappaletta 400 MHz prosessoreita. Vaikka skaalautuminen olikin sunin koneessa lähes ideaalista, hävisi se suorituskyvyssä alpha prosessoreille reilusti myös prosessorien määrällä 1-4.



Kuva 6: Molekyylidynamiikkasimulaation skaalautuvuus eri ongelman koilla.

Beowulf-klusterissa skaalautuvuuden määrää ennen kaikkea prosessorien välisen kommunikaation määrä. Skaalautuvuutta voi yrittää parantaa valitsemalla ison ongelman koon, jolloin laskennan määrää suhteessa kommunikaation määrään saadaan kasvatettua.

## 6. Yhteenveto

Beowulf-klusteri soveltuu monenlaisten tieteellisten ongelmien ratkaisuun. Sen skaalautuvuus määräytyy ensisijaisesti sovelluksen käyttämän kommunikaation määrän perusteella. Juuri hitaan verkkoratkaisunsa takia Beowulfista ei vielä ole syrjäyttämään

perinteisiä supertietokoneita kuten Cray T3E:tä. Kuitenkin huolellisella suunnittelulla ja hyvin rinnakkaistuvalla ongelmalla Beowulfilla on mahdollista saavuttaa perinteisiä supertietokoneita vastaava skaalautuvuus. Beowulf-klusterin rakentaminen on tällä hetkellä ehdottomasti halvin tapa hankkia supertietokone.

## 7. Lähteet

- [1] **ASCI Red UltraComputer**,  
<http://www.sandia.gov/ASCI/Red.htm> (29.4.1999)
- [2] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer. **BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION**, Proceedings, International Conference on Parallel Processing, 95. <http://www.beowulf.org/papers/ICPP95/icpp95.ps> (29.4.1999)
- [3] Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling Becker, Phillip Merkey. **Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs**, Proceedings, IEEE Aerospace, 1997. <http://www.beowulf.org/papers/AA97/aa97.ps>
- [4] Jack Dongarra, Hans Meuer, Erich Strohmaier. **TOP500 Supercomputer Sites**, (lokakuu 1998) [http://www.top500.org/lists/1998/11/top500\\_9811.ps](http://www.top500.org/lists/1998/11/top500_9811.ps) (29.4.1999)
- [5] **Science Research on Avalon**,  
<http://cnls.lanl.gov/avalon/science.html> (29.4.1999)
- [6] Daniel Katz. **Beowulf Applications and User Experiences**,  
<http://www.cacr.caltech.edu/beowulf/tutorial/apps2.pdf> (29.4.1999)
- [7] **SGI - Cray T3E White Papers: Performance of the Cray T3E Multiprocessor**,  
<http://www.sgi.com/t3e/performance.html> (29.4.1999)
- [8] **Myrinet Performance Measurements**,  
<http://www.myri.com/myrinet/performance/index.html> (29.4.1999)
- [9] Cristopher Bohn. **Asymmetric Load Balancing on a Heterogeneous Cluster of PCs (DRAFT)**, [http://members.aol.com/EngrBohn/thesis/ge99m02\\_draft.pdf](http://members.aol.com/EngrBohn/thesis/ge99m02_draft.pdf) (29.4.1999)

- [10] **MPI – The Message Passing Interface Standard,**  
<http://www.mpi-forum.org/> (29.4.1999)
- [11] Juha Haataja, Kaj Mustikkamäki. **Rinnakkaisohjelmointi MPI:llä**
- [12] Jussi Rahola. Seminaariesitelmä 25.3.1999: **MPI: Message Passing Interface,**  
[http://www.lce.hut.fi/teaching/S-114.240/esitykset/Jussi\\_Rahola.ps.gz](http://www.lce.hut.fi/teaching/S-114.240/esitykset/Jussi_Rahola.ps.gz) (30.4.1999)
- [13] **MPICH – A Portable MPI Implementation,**  
<http://www-unix.mcs.anl.gov/mpi/mpich/> (29.4.1999)
- [14] **Message Passing Interface (MPI) FAQ,**  
<http://www.erc.msstate.edu/mpi/mpi-faq.html> (29.4.1999)
- [15] **ScaLAPACK Project,**  
<http://www.netlib.org/scalapack/> (5.5.1999)
- [16] **PETSc Home Page,**  
<http://www-unix.mcs.anl.gov/petsc/petsc.html> (5.5.1999)
- [17] M. S. Warren, T. C. Germann, P. S. Lomdahl, D. M. Beazley and J. K. Salmon.  
**Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for \$150k,.**  
*Supercomputing '98*, Los Alamitos, 1998. IEEE Comp. Soc.  
<http://loki-www.lanl.gov/papers/sc98/avalon.ps> (5.5.1999)
- [18] **The NAS Parallel Benchmark,**  
<http://science.nas.nasa.gov/Software/NPB/> (5.5.1999)
- [19] **M-VIA FAQ,**  
<http://www.nersc.gov/research/FTG/via/faq.html> (5.5.1999)