Ytelsesmålinger gjort på DALTON

A Report for the NOTUR Project *Emerging Technologies: Cluster*

Daniel Stødle
Otto J. Anshus, John Markus Bjørndalen
Department of Computer Science,
University of Tromsø

1.0 Innledning

"High Performance Distributed Computing"-gruppen på IfI i Tromsø har gjort en preliminær analyse av ytelsen til ulike tunge, parallelliserte applikasjoner. En av disse applikasjonene var Dalton, et program for å utføre ulike kvantekjemiske beregninger.

Ytelsen til Dalton ble målt ved forskjellige problemstørrelser og på flere forskjellige klynger av datamaskiner med forskjellige arkitekturer i den hensikt å skaffe tilveie et første grunnlag for å analysere om det var mulig å forbedre ytelsen og hvordan dette i så fall kunne gjøres.

Det er ikke en del av denne rapporten å vurdere effekten av å bruke de verktøy (PastSet, PARHS, EventSpace) vi har utviklet for å rekonfigurere parallelle eregninger.

Dalton er et stort og komplisert program, og det er til nå kun deler av Dalton som er parallellisert. Disse delene benytter enten MPI eller PVM, og den delen vi målte ytelsen til benyttet seg av en enkel master-slave modell, og MPI.

Mer informasjon, inkludert en forklaring av master-slave tilnærmingen brukt i Dalton, kan finnes i vedlagte poster om Dalton presentert på NOTUR Gathering 2003 (Appendix).

2.0 Mål

På kort og litt lengere sikt ønsker vi å:

- Dokumentere hvordan Dalton skalerer (oppfører seg ytelsesmessig og ressursmessig) når problemstørrelse og antall noder det utføres på øker.
- Identifisere hvilke trekk det er ved Dalton som Dalton skalerer eller ikke.
- Identifisere måter å forbedre ytelsen på.

3.0 Ytelsesmålinger

Vi har til nå målt kjøretiden til Dalton med følgende faktorer og parametre:

- Tre forskjellige cluster ble brukt. Hver klynge hadde forskjellig arkitektur, og ytelse (prosessor, internlager, busser, I/O)
    - "Snowstorm" Itanium cluster, IT avd., UiT: Samling av 2 og 4 veis Itanium 2 noder med totalt 22 prosessorer.
        - Seven HP rx2600 servers, each with 2 CPU's: 900 MHz Itanium 2, Cache, L1/L2/L3: 32kB / 256kB / 1.5MB (all are on-chip), 2 GB of DDR-SDRAM memory, 2 x 36 GB disks
        - Two HP rx5670 servers, each with 4 CPUs: 1 GHz Itanium 2 , Cache, L1/L2/L3: 32kB / 256kB / 3.0MB (all are on-chip), 4 GB of DDR-SDRAM memory, 4 x 36 GB disks
        - Red Hat Linux Advanced Server, 64 bits
        - Nodene er sammenkoblet med Gigabit Ethernet.
        - Antall prosessorer brukt ble variert fra 1 til 22.
    - Four way (4W), IFI, UiT
        - Eight nodes with four Pentium Pro 166MHz processors sharing 128MB physical memory, 512KB L2 cache
        - Nodene er sammenkoblet med 100MHz Ethernet
        - Antall prosessorer brukt ble variert fra 1 til 32.
    - Eight way (8W), IFI, UiT
        - Fire noder, hver med åtte Pentium Pro 200MHz prosessorer som fysisk deler 2GB internlager
        - Nodene er sammenkoblet med 100MHz Ethernet
        - Antall prosessorer brukt ble variert fra 1 til 32.
- Vi ga Dalton to forskjellige input-data av varierende kompleksitet.

4.0 Metodologi

Vi målte totale regnetid for sekvensiell versjon av Dalton på en prosessor for hver klynge, og for den parallelle versjonen.

Vi brukte et profileringsbiblioktek til MPI, for å studere Daltons kommunikasjonsmønster, og om det var tilfeller der slavenoder sto i lengre tid og ventet på mer aa gjøre.

Tilslutt utførte vi tester for å avgjøre om master-noden kunne komme til å utgjøre en flaskehals.

5.0 Resultater

Dalton skalerer relativt bra. Med 32 noder observerte vi en speedup på mellom 25 og 32, sammenlignet med den sekvensielle versjonen. Det er dog viktig å vare oppmerksom på at dette kun er gyldig når den sekvensielle versjonen kjører uten å cache midlertidige resultater. Dersom sekvensielle Dalton cacher sine midlertidige resultater, er den bare 3-5 ganger tregere enn den parallelle utgaven på 32 noder.

Til tross for at 8-veis clusteret ikke hadde lokale disker (kun et nettverksmontert filsystem), var den sekvensielle Dalton fremdeles betydelig raskere. Dette indikerer at nettverksbåndbredde ikke nødvendigvis vil bli et problem, dersom man innfører caching i den parallelle utgaven.

Når det gjelder Daltons kommunikasjonsmønster, er det ingen overraskelser. Mønsteret er typisk for master-slave "bag-of-tasks" orienterte programmer, med lite kommunikasjon, lite synkronisering og generelt god utnyttelse av slavenodene. Masternoden gjør relativt lite arbeid, og er blokkert mesteparten av tiden.

Tilslutt undersøkte vi om masternoden kunne være en potensiell flaskehals. Det ga ingen forskjeller i regnetiden om masternoden var på sen eller rask hardware i forhold til slavenodene. Vi prøvde både mønstret “master rask-slaver sen” og “master sen-slaver rask”. Dette ble dog kun testet opp til 32 noder, og det er mulig at høyere antall noder kan føre til at masteren blir overbelastet og begrenser hvor raskt Dalton kan kjøre.

6.0 Konklusjon

Dalton skalerer bra i forhold til seg selv når antall noder øker fra 2 til 32. Men 32 noder kun mellom 3-5 ganger raskere enn sekvensielle Dalton med caching. Dette er fordi den parallelle versjonen ikke cacher sine midlertidige resultater. Her er det potensialer for å forbedre ytelsen ved at de midlertidige resultatene caches parallelle Dalton.

Caching i parallell Dalton vil gi utfordringer å løse i at diskplass må deles mellom nodene, samt at ulike "jobber" må få en affinitet for en av nodene, slik at kommunikasjon/datautveksling kan reduseres til et minimum.

Masteren utgjør ingen flaskehals i våre eksperiment, og den kan kjøres på senere (eldre, billigere) hardware enn resten av nodene.

7.0 Takk

Takk til Kenneth Ruud, Kjemi, for verdifull hjelp og innsikt i en applikasjon vi aldri før har sett på, og til Roy Dragseth, IT avd, for support på Itanium klyngen til Universitetet i Tromsø.

Appendix:

Dalton Poster at NOTUR Gathering, Oslo, 2003

NOTUR Emerging

Measuring the Performance

- By Daniel Stødle, Otto J. Anshus and John Markus Bjørndalen, IFI, UiTø

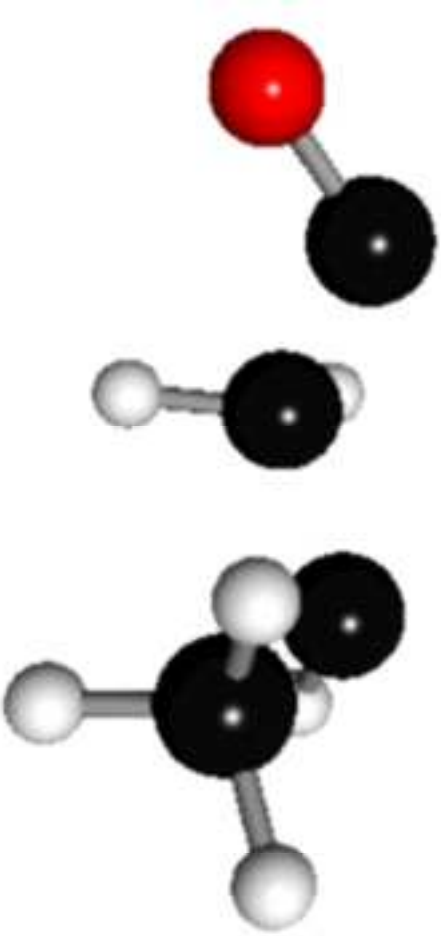
- Thanks to Kenneth Ruud for very valuable Dalton support and Roy Dragseth for his assistance on the Snowstorm cluster

Technologies

Cluster Computing

Parallel of Dalton [1]

- **In cooperation with the HPC unit at IT-avd, UiTø and Anne Cathrine Elster, NTNU.**



John Dalton (1766-1844)
The founder of modern atomic theory

What is Dalton?

- **An application for performing quantum chemistry computations**
- **Distributed to over 750 different research institutions worldwide**
- **Runs on a wide range of platforms**
- **Dalton is available free of charge**
- **More information is available from http://www.kjemi.uio.no/software/dalton/**

Our objectives

- **Study the parallel version of Dalton**

- **We want to know:**

  - How does it scale?

  - Why does it scale?

  - How can the performance be optimized?

  - What other factors influence Dalton's performance?

  - Can we improve Dalton's performance?

Clusters used

- Used three different clusters
  - Snowstorm, a mixed 2- and 4-way Itanium 2 cluster with a total of 22 CPUs
  - 4-way cluster with a total of 32 CPUs
  - 8-way cluster, totalling 32 CPUs
- Snowstorm is brand new, the two others are becoming dated
- Cluster interconnect is ethernet
- 8-way cluster has no local disks, but uses NFS for storage and temporary files

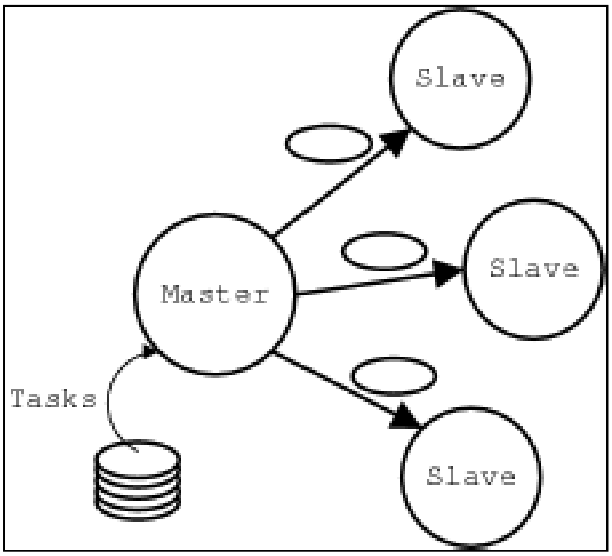
Experiment details

- **Benchmarks on 1-32 CPUs**
- **Two different benchmarks of varying complexity**
- **Second experiment to determine how the master affects the computation**
- **Third experiment to profile communication patterns**
  - **How much communication takes place?**
  - **Opportunities for optimization?**

About Parallel Dalton

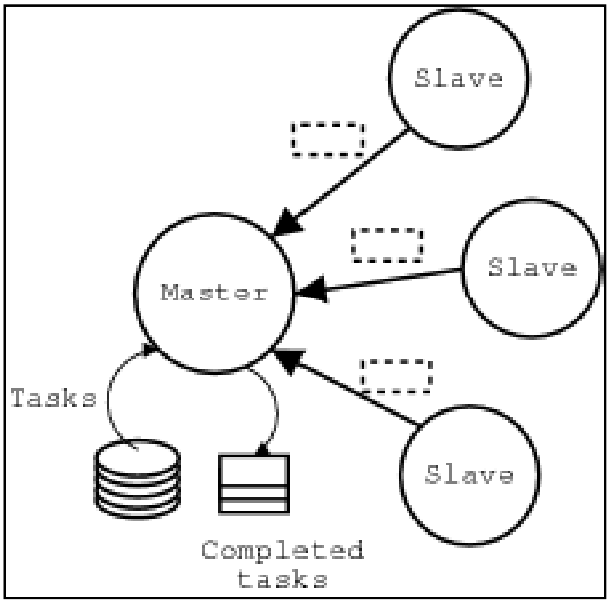
- **Calculations are parallelized using the Master-Slave paradigm**

- **Bag-of-tasks to distribute the computation:**



1. The master distributes tasks to each slave. The tasks may be of varying size and complexity, but efforts are made to make the load spread evenly across the nodes.

2. As the slaves complete their work units, they return their results to the master. The master then provides the slaves with new work units, until the entire calculation completes.

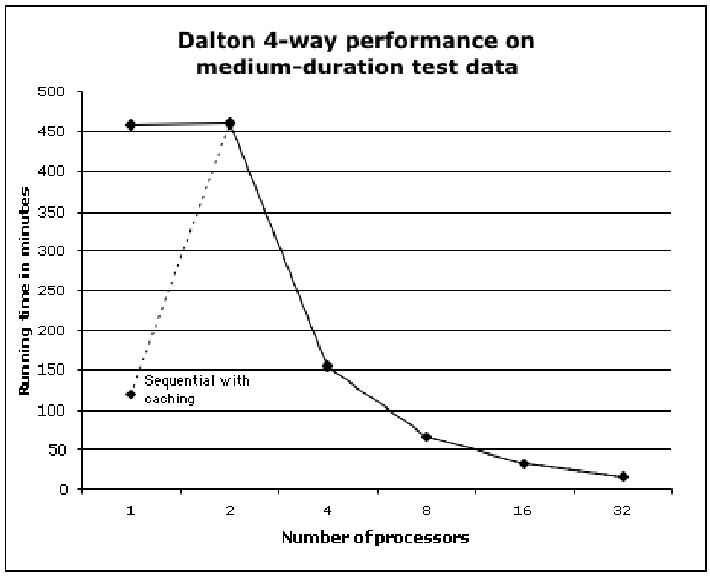


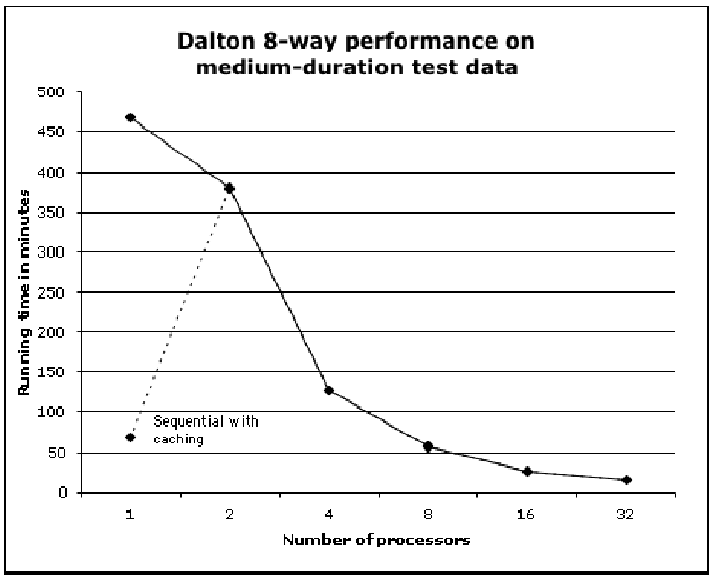
- **Parallel Dalton supports both MPI and PVM**

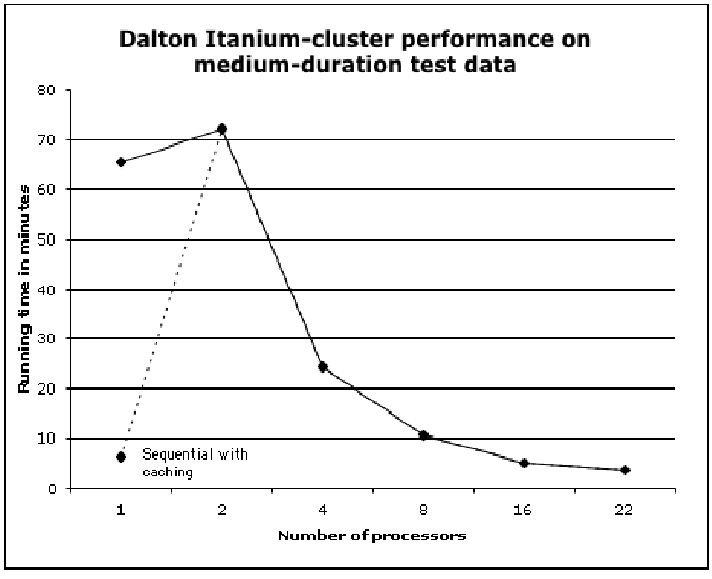
- **Only one master node (no hierarchies)**

Performance graphs

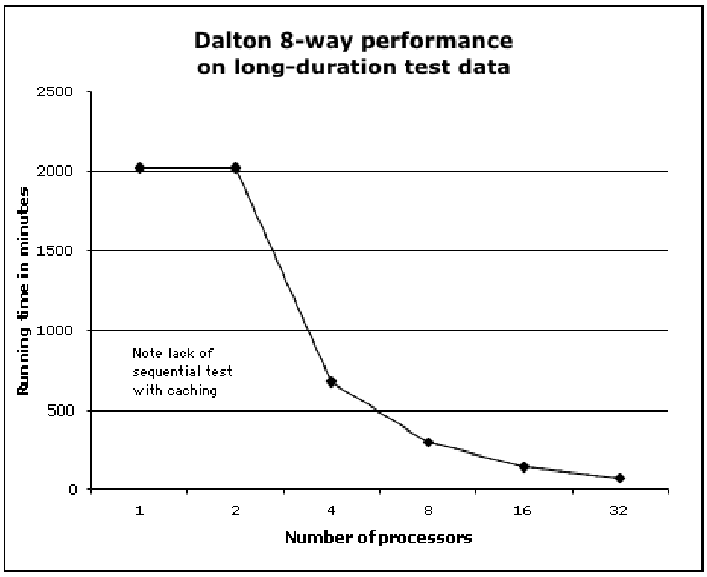
- 4-way, 8-way and Itanium cluster performance

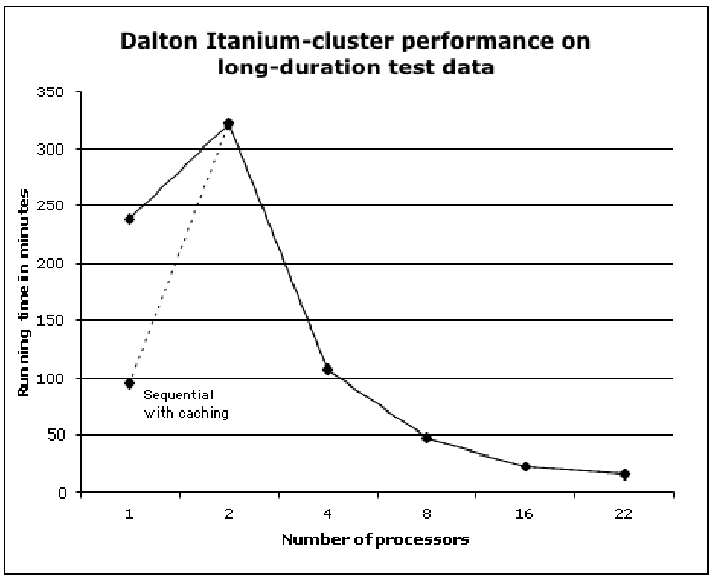






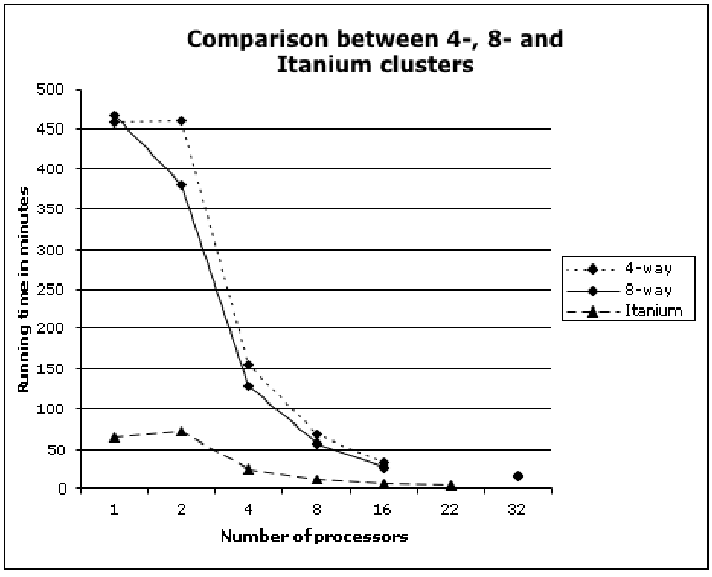
of the benchmarks





- 8-way and Itanium cluster performance on complex test data

- Performance comparison

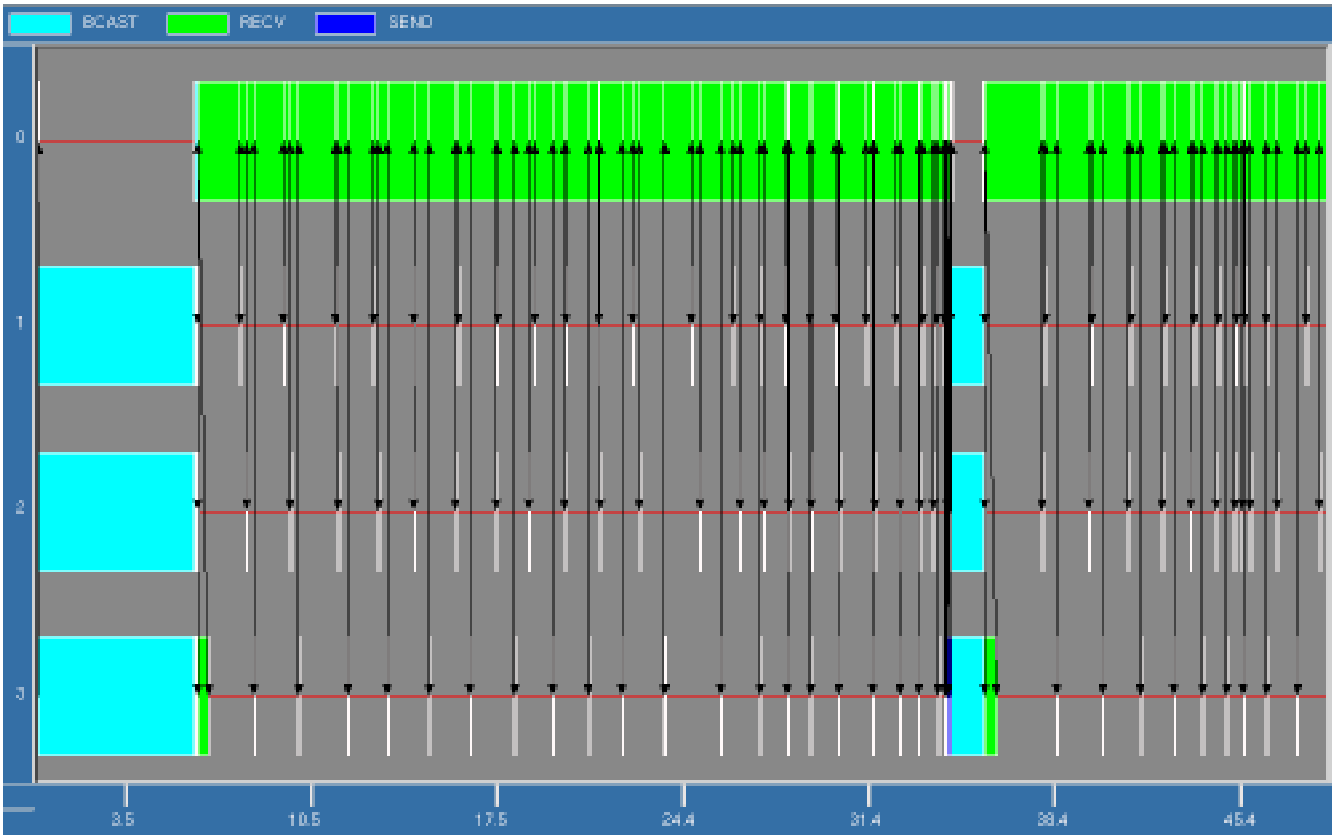


Preliminary results

- **Good speed-up**
  - **32 nodes is between 25 and 32 times faster than 1 CPU without caching**
- **Important tradeoff between disk usage and speed**
  - **Parallel version doesn't cache any partial results**
  - **Sequential version can cache partial results, but disk usage increases quickly**
- **Sequential Dalton with caching is only 3-5 times slower than parallel Dalton on 32 CPUs**
- **8-way cluster only had network storage, yet sequential Dalton with caching is still substantially faster than the non-caching version**

Communications

- The figure shows the pattern of communication between Dalton's nodes
- Pattern is typical for bag-of-tasks oriented parallel programs

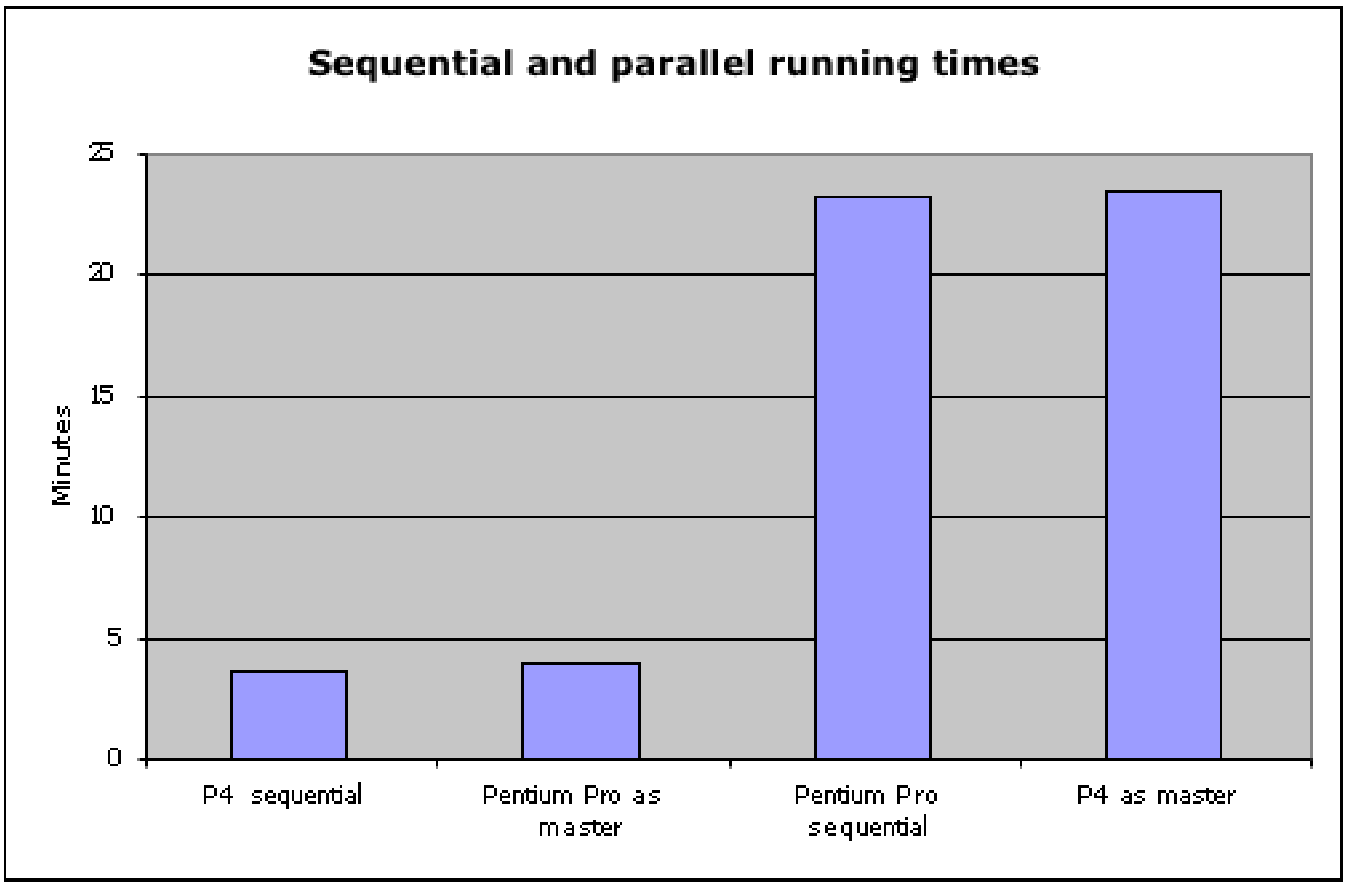


- Little communication
- Master is blocked most of the time
- Work done by master is limited

and bottlenecks

- Graph shows how going from sequential (w/o caching) to two nodes affects the computation

  - Will master on a slower node affect the general performance of the computation, or is the master able to keep up?



- Sequential running time and running time with one master and one slave are almost equal

  - Master does not add considerable overhead

Analysis & conclusions

- **Dalton scales very well up to 32 nodes**
  - **Master might become a problem on larger clusters**
- **Caching partial results boosts performance**
  - **But does it scale to larger problems?**
- **Master does not appear to be a bottleneck**
  - **Master is generally blocked, waiting to send tasks to its slaves**
- **Best way to increase performance is to add caching to the parallel version**
  - **Must find a way to share disk space over multiple nodes**
  - **Must give tasks an affinity for the node on which the task last ran**

Future work

- **Use PATHS to add caching capabilites to Dalton**
  - **PATHS also allows us to reconfigure the placement of data on each node, making optimization work simpler**
- **Find Dalton's typical cache size, when used on real problems**
- **Look at the performance of the parallel matrix diagonalizing code**
  - **Find bottlenecks**
  - **Determine opportunities for optimization**

1) Dalton, a molecular electronic structure program, Release 1.2 (2001), written by T. Helgaker, H.J.Aa. Jensen, P. Jørgensen, J. Olsen, K. Ruud, H. Ågren, A.A. Auer, K.L. Bak, V. Bakken, O. Christiansen, S. Coriani, P. Dahle, E.K. Dalskov, T. Enevoldsen, B. Fernandez, C. Hättig, K. Hald, A. Halkier, H. Heiberg, H. Hettema, D. Jonsson, S. Kirpekar, R. Kobayashi, H. Koch, K.V. Mikkelsen, P. Norman, M.J. Packer, T.B. Pedersen, T.A. Ruden, A. Sanchez, T. Saue, S.P.A. Sauer, B. Schimmelpfennig, K. O. Sylvester-Hvid, P.R. Taylor, and O. Vahtras