Optimization on the Computational Grid

JEFF LINDEROTH
ISE Department
COR@L Lab
Lehigh University
jt13@lehigh.edu

Enterprise-Wide Optimization (EWO) Tele-Seminar
Allentown, PA
May 9, 2006
An Abject Apology

- Much of this work was done long ago
  - There is *no* original material!
  - I even have reused many of the same hackneyed jokes
- I have a good excuse
My Good Excuse—Jacob Linderoth
Outline

- What I lack in quality, I make up for in quantity: $\geq 70$ slides
- What is The Computational Grid?
- Tools for using the Computational Grid
  - Condor and MW
- Solving large optimization problem instances on the Computational Grid
  - Two-stage stochastic linear programming
  - Quadratic Assignment Problem

Theme of the Talk
Fit your algorithm to your computational platform!
Come on Let’s Play the Feud

‘‘100 People Surveyed. Top 5 answers are on the board. Here’s the question...’’

Name one common use of the Internet
The Big Board

1. email
2. Looking up answers to homework problems
3. Looking up stock quotes
4. Downloading MP3s
5. Looking at pictures of Anna Kournikova
Strike!

- Doing Computations
People envision a “Computational Grid” much like the national power grid:

- Users can seamlessly draw computational power whenever they need it.
- Many resources can be brought together to solve very large problems.
- Gives application experts the ability to solve problems of unprecedented scope and complexity, or to study problems which they otherwise would not.
This ain’t easy!

- User access and security
  - Who should be allowed to tap in?
- Interfaces
  - How should they tap in?
- Heterogeneity
  - Different hardware, operating systems, and software
- Dynamic
  - Participating Grid resources may come and go
  - Fault-Tolerance is very important!
- Communicationally challenged
  - Machines may be very far apart ⇒ slow communication.
Building a Grid

- There have been *lots* of software tools that accomplish these goals to varying degrees.
- One problem remains: **GREED!**
  - Most people don’t want to contribute “their” machine!
- How to induce people to contribute their machine to the Grid?
  - Screensaver – *seti@home*
  - Social Welfare – *fightaids@home*
  - Offer frequent flyer miles – company went bankrupt
  - Let the people *keep control* over their machine
  - Give donaters a chance to use the Grid
Tool I – Condor

Peter Keller
Miron Livny
Erik Paulsen
Rajesh Raman
Marvin Solomon
Todd Tannenbaum
Doug Thain
Derek Wright

http://www.cs.wisc.edu/condor
What is Condor?

- Manages collections of “distributively owned” workstations
  - User need not have an account or access to the machine
  - Workstation owner specifies conditions under which jobs are allowed to run
  - All jobs are scheduled and “fairly” allocated among the pool
- How does it do this?
  - Scheduling/Matchmaking
  - Jobs can be checkpointed and migrated
  - Remote system calls provide the originating machines environment
## Matchmaking

<table>
<thead>
<tr>
<th>MyType</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetType</td>
<td>Machine</td>
</tr>
<tr>
<td>Owner</td>
<td>ralphs</td>
</tr>
<tr>
<td>Cmd</td>
<td>cplex</td>
</tr>
<tr>
<td>Args</td>
<td>seymour.d10.mps</td>
</tr>
<tr>
<td>HasCplex</td>
<td>TRUE</td>
</tr>
<tr>
<td>Memory</td>
<td>≥ 64</td>
</tr>
<tr>
<td>Rank</td>
<td>KFlops</td>
</tr>
<tr>
<td>Arch</td>
<td>SUN4u</td>
</tr>
<tr>
<td>OpSys</td>
<td>SOLARIS27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MyType</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetType</td>
<td>Job</td>
</tr>
<tr>
<td>Name</td>
<td>nova9</td>
</tr>
<tr>
<td>HasCplex</td>
<td>TRUE</td>
</tr>
<tr>
<td>Arch</td>
<td>SUN4u</td>
</tr>
<tr>
<td>OpSys</td>
<td>SOLARIS27</td>
</tr>
<tr>
<td>Memory</td>
<td>256</td>
</tr>
<tr>
<td>KFlops</td>
<td>53997</td>
</tr>
<tr>
<td>RebootedDaily</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Checkpointing/Migration

Professor Arrives
5 min

Grad Student Arrives
5 min

8:10am
12pm
Other Condor Features

- **Pecking Order**
  - Users are assigned priorities based on the number of CPU cycles they have recently used.
  - If someone with higher priority wants a machine, your job will be booted off.

- **Flocking**
  - Condor jobs can negotiate to run in other Condor pools.

- **Glide-in**
  - Globus (Grid computing toolkit from Argonne) provides a “front-end” to many traditional supercomputing sites.
  - Submit a Globus job which creates a temporary Condor pool on the supercomputer, on which users jobs may run.
Condor is Really Taking Off

Companies Presenting at Recent Condor Week 2006

- JP Morgan
- Altera Corporation
- Hartford Life
- Micron
- UBS
- Yahoo

- http://www.cs.wuscs.edu/condor
Grid computing applications

**What Condor is good at...**
- Naturally/Pleasantly/Embarrassingly Parallel Applications
- The computation can *a priori* be broken into tasks
  - Monte Carlo Methods, Code Cracking
  - seti@home – the search for intelligent life

**What Condor is (not so) good at...**
- Parallel Applications with non-trivial control structures
  - Like Optimization Algorithms for EWO!

- To make parallel algorithms dynamically adjustable and fault-tolerant, we could use the master-worker paradigm
- What is the master-worker paradigm, you ask?
Master-Worker!

- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Fault-tolerant
- Dynamic
Tool II – MW

Wen-Han Goh
Sanjeev Kulkarni
Greg Thain
Mike Yoder

Jean-Pierre Gouix
Jeff Linderoth

http://www.cs.wisc.edu/condor/mw
There are three abstraction is the master-worker paradigm: Master, Worker, and Task.

MW is a software package that encapsulates these abstractions
  - C++ abstract classes
  - User writes 10 functions
  - The MWized code will transparently adapt to the dynamic and heterogeneous environment

MW also has abstract layer to resource management and communications packages
  - Condor/PVM, Condor/Files
  - Condor/Unix Sockets
  - Single processor
MW Classes

- **MWMaster**
  - getuserinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()

- **MWTask**
  - (un)pack_work
  - (un)pack_result

- **MWWorker**
  - unpack_worker_init_data()
  - execute_task()
MWApplications

- MWFATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWSVM (Ferris, Voelker) – A column generation code for solving support vector machine problems
- MWQPBB (Linderoth) – A branch-and-bound code for nonconvex quadratic programming
- MWAND (Linderoth, Shen) – A (preliminary) nested-decomposition code for multistage stochastic linear programming
- MWLShaped/MWATR (Linderoth, Shapiro, Wright) – A (trust-region) cutting plane code for linear stochastic programming and verification of solution quality
- MWQAP (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem
Stochastic Programming Collaborators

Alex Shapiro
ISyE
Georgia Tech

Steve Wright
Computer Science Department
University of Wisconsin-Madison
A Stochastic Program

\[ \min_{x \in X} \{ f(x) \} \overset{\text{def}}{=} \mathbb{E}_P F(x, \omega) \]

- \( F(x, \omega) \) is a real-valued function of two vector variables.
- \( x \in \mathbb{R}^n \) and \( \omega \in \mathbb{R}^d \)
- \( \omega \) is a random vector having probability distribution \( P \)
- \( X \subseteq \mathbb{R}^n \)

A Two-Stage Stochastic LP

\[ \min_{x \geq 0, Ax = b} c^T x + Q(x) \]

- \( Q(x) \) def \( \mathbb{E}_P [Q(x, \omega)] \)
- The recourse problem:
  \[ Q(x, \omega) \overset{\text{def}}{=} \min q^T y \]
  \[ W y = h(\omega) - T(\omega)x \]
  \[ y \geq 0 \]
Best-Known Solution Procedure

METHO
Two-Stage Stochastic Linear Programming

- We assume that the $P$ has finite support, so $\omega$ has a finite number of possible realizations (*scenarios*):

$$Q(x) = \sum_{i=1}^{N} p_i Q(x, \omega_i)$$

- For a partition of the $N$ scenarios into chunks $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_t$, let $Q[j](x)$ be the contribution of the $j$th chunk to $Q(x)$:

$$Q[j](x) \overset{\text{def}}{=} \sum_{i \in \mathcal{N}_j} p_i Q(x, \omega_i)$$

- so then $Q(x) = \sum_{j=1}^{t} Q[j]$
Important (and well-known) Facts

- \( Q(x, \omega_i) \), \( Q_[.] (x) \), and \( Q(x) \) are piecewise linear convex functions of \( x \).
- If \( \pi_i \) is an optimal dual solution to the linear program corresponding to \( Q(\hat{x}, \omega_i) \), then \( -T^T_i \pi_i \in \partial Q(\hat{x}, \omega_i) \)
  - \( g_j(\hat{x}) \equiv \sum_{i \in \mathcal{N}_j} -p_i T^T_i \pi_i \in \partial Q[.] (\hat{x}). \)
- Represent \( Q[.] (x) \) by an artificial variable \( \theta_j \) and find supporting planes for \( \theta_j \)
  - \( \theta_j \geq g_j(x^k)^T x + (Q[.] (x^k) - g_j^T x^k) \) \( (*) \)
- Evaluation of \( Q(\hat{x}) \) is separable
- We can solve linear programs corresponding to each \( Q(\hat{x}, \omega_i) \) independently – in parallel!
Worth 1000 Words

\[ Q(x) \]

\[ x \]
Worth 1000 Words

\[ Q(x) \]

\[ x \quad x^k \]
Worth 1000 Words

\[ Q(x) \]
Multicut L-shaped method

1. Solve the **master problem** $M$ with the current $\theta_j$-approximations to $Q_{[j]}(x)$ for $x^k$.

2. Solve the **subproblems**, $(s_j)$ evaluating $Q_{[j]}(x^k)$ and obtaining a subgradient $g_j(x^k)$. Add inequalities (*) to the master problem.

3. $k = k + 1$. Goto 1.
A Trust Region Method

- The LShaped method suffers from “stability” problems,
  - Especially in early iterations when a “good enough” model of $Q(x)$ is not known
  - Especially bad if started from a good guess at the solution
- 🤔 Borrow the trust region concept from NLP 🤔
  - At iteration $k$, impose constraints $\|x - x^k\|_\infty \leq \Delta_k$
  - Accept new iterate if it improves the objective by a “sufficient” amount. Potentially increase $\Delta_k$.
  - Otherwise, improve the estimation of $Q(x^k)$, resolve master problem, and potentially reduce $\Delta_k$. 
Stop Thief!

“Lesser artists borrow, great artists steal.”

– Igor Stravinsky

- Stabilizing Bender’s decomposition is hardly a new idea
  - Marsten, Hogan, Blankenship (’75)
  - Lemaréchal (’75, …)
  - Kiwi (’83, …)
  - Ruszczyński (’86)
  - Neame, Boland, and Ralph (’98)
**minor Differences**

- We like $\| \cdot \|_\infty$ trust region
  - The master problem is just an LP with modified bounds
  - Degeneracy has not reared its ugly head
  - (Perhaps due to better LP solvers)
- Convergence analysis is somewhat different
  - Convergence for arbitrary subgradients – not necessarily extreme point dual solutions $\pi(\omega_i)$.
- Liberal cut deletion from master is possible
  - This is *especially* important in a grid computing implementation
Preparing for the Grid

- **Work**
  - One or more scenario chunks $\mathcal{N}_{j_1}, \ldots \mathcal{N}_{j_C}$ and point ($\hat{x}$)

- **Result**
  - A subgradient of each of the $Q_{[j_k]}(\hat{x})$.

- **act_on_completed_task**
  - Add subgradient inequalities to master problem
  - Solve master problem if all workers have reported their results for the iteration
Headaches!

- Solving the master problem is a “synchronization point” of the algorithm
  - Amdahl’s Law: Parallel efficiency is limited by the amount of synchronization.

- This synchronization problem is **MUCH** worse in Computational Grid computing environments!
Worker Usage—Number of Idle Workers
Stamp Out Synchronicity!

- On a Grid, different processors act at different speeds,
- Many may wait idle for the “slowpoke”
- Even worse, grid computing tools can fail to inform the user that their worker has failed!

Asynchronicity is key!

Asynchronous methods are preferred for traditional parallel computing environments. They are nearly *required* for Grid Computing environments!
ATR – An Asynchronous Trust Region Method

- Keep a “basket” $B$ of trial points for which we are evaluating the objective function.
- Make decision on whether or accept new iterate $x^{k+1}$ after entire $Q(x^k)$ is computed.
- Populate the basket quickly by initially solving the master problem after only $\alpha\%$ of the scenario LPs have been solved.
- Greatly reduces the synchronicity requirements.
- Might be doing some “unnecessary” work – the candidate points might be better if you waited for complete information from the preceding iterations.
The World’s Largest LP

- Storm – A stochastic cargo-flight scheduling problem (Mulvey and Ruszczyński)
- We aim to solve an instance with 10,000,000 scenarios
- \( x \in \mathbb{R}^{121}, y_k \in \mathbb{R}^{1259} \)
- The deterministic equivalent LP is of size

\[ A \in \mathbb{R}^{985,032,889 \times 12,590,000,121} \]
## The Super Storm Computer

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>184</td>
<td>Intel/Linux</td>
<td>Argonne</td>
</tr>
<tr>
<td>254</td>
<td>Intel/Linux</td>
<td>New Mexico</td>
</tr>
<tr>
<td>36</td>
<td>Intel/Linux</td>
<td>NCSA</td>
</tr>
<tr>
<td>265</td>
<td>Intel/Linux</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>88</td>
<td>Intel/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>239</td>
<td>Sun/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>124</td>
<td>Intel/Linux</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>90</td>
<td>Intel/Solaris</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>13</td>
<td>Sun/Solaris</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>9</td>
<td>Intel/Linux</td>
<td>Columbia U.</td>
</tr>
<tr>
<td>10</td>
<td>Sun/Solaris</td>
<td>Columbia U.</td>
</tr>
<tr>
<td>33</td>
<td>Intel/Linux</td>
<td>Italy (INFN)</td>
</tr>
<tr>
<td>1345</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Statistics from Storm Computation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall clock time</td>
<td>31:53:37</td>
</tr>
<tr>
<td>CPU time</td>
<td>1.03 Years</td>
</tr>
<tr>
<td>Avg. # machines</td>
<td>433</td>
</tr>
<tr>
<td>Max # machines</td>
<td>556</td>
</tr>
<tr>
<td>Parallel Efficiency</td>
<td>67%</td>
</tr>
<tr>
<td>Master iterations</td>
<td>199</td>
</tr>
<tr>
<td>CPU Time solving the master problem</td>
<td>1:54:37</td>
</tr>
<tr>
<td>Maximum number of rows in master problem</td>
<td>39647</td>
</tr>
</tbody>
</table>

---

**TA-DA!!!!!!**
Number of Workers

![Graph showing the number of workers over time](image-url)
The Quadratic Assignment Problem

Mathematical Formulation

\[
\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)}
\]

- Assign facilities to locations
- QAP is NP-"Super"-Hard
- Branch and Bound is the method of choice, but very few tight, computable, bounds exist.
QAP Collaborators

\{ \textbf{Kurt Anstreicher} \\
University of Iowa \}

\{ \textbf{Nate Brixius} \\
Micro\$oft \}

\{ \textbf{Jean-Pierre Goux} \\
Argonne, Northwestern, and Artelys \}
How to solve BIG problems with Branch and Bound

1. Start with a state of the art bounding technique
2. Do intelligent search and branching
   - We employ/extend “strong branching” in this non-linear context
   - Use different branching strategies depending on the relative difficulty of a node
3. Pick a very powerful computing platform
   - The Computational Grid!
4. Fit the algorithm to the platform
Kurt Anstreicher and Nate Brixius showed that the solution to the following problem provides a lower bound to the solution of QAP:

$$\min f(X) \equiv \text{vec}(X)^T Q \text{ vec}X + C \cdot X$$

such that

$$X e = X^T e = e, \quad X \geq 0.$$ 

$$Q \equiv (B \otimes A) - (I \otimes S) - (T \otimes I)$$ 

$S$ and $T$ are obtained from the spectral decompositions of $A$ and $B$ 

There are more details that I don’t understand 

This is a **convex** quadratic programming problem relaxation
An Oldie but a Goodie

The Frank-Wolfe algorithm is used to solve the quadratic programming subproblems

1. \( X^*_k = \arg\min_{X \in \Pi} \left\{ \nabla f(X_k) \cdot X \right\} \). Linear Assignment Problem

2. \( X_{k+1} = X_k + \alpha (X^*_k - X_k) \), where \( 0 \leq \alpha \leq 1 \) minimizes \( f(X_{k+1}) \). Closed form solution

3. Go to 1.

Each \( X^*_k \) comes with a matrix of “reduced costs”, from which a valid lower bound can be deduced: \( z_{OPT} \geq f(X_k) - U_k \cdot X_k \).
How to solve BIG problems with Branch and Bound

1. Start with a state of the art bounding technique
2. Do intelligent search and branching
   - We employ/extend “strong branching” in this non-linear context
   - Use different branching strategies depending on the relative difficulty of a node
3. Pick a very powerful computing platform
   - The Computational Grid!
4. Fit the algorithm to the platform
We do not do "normal" IP branching

Instead do \( n \)-ary branching
- Fix each assignment in a row or column

Also adapted \textit{strong-branching} to this nonlinear context.
Strong Branching

- There are many “unsatisfied entities”, how do we decide on which to branch?
  - We would like to select an entity that will increase the bound as much as possible
- MIP – Tentatively select a fractional variable, and perform a small number of dual simplex pivots
- QAP – Tentatively select a row or column and perform a small number of Frank-Wolfe iterations
- Branch on entity where the lower bounds improve “the most”
How to solve BIG problems with Branch and Bound

1. Start with a state of the art bounding technique
2. Do intelligent search and branching
   - We employ/extend “strong branching” in this non-linear context
   - Use different branching strategies depending on the relative difficulty of a node
3. Pick a very powerful computing platform
   - The Computational Grid!
4. Fit the algorithm to the platform
### Our Computational Grid

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>414</td>
<td>Intel/Linux</td>
<td>Argonne</td>
</tr>
<tr>
<td>96</td>
<td>SGI/Irix</td>
<td>Argonne</td>
</tr>
<tr>
<td>1024</td>
<td>SGI/Irix</td>
<td>NCSA</td>
</tr>
<tr>
<td>16</td>
<td>Intel/Linux</td>
<td>NCSA</td>
</tr>
<tr>
<td>45</td>
<td>SGI/Irix</td>
<td>NCSA</td>
</tr>
<tr>
<td>246</td>
<td>Intel/Linux</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>146</td>
<td>Intel/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>133</td>
<td>Sun/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>190</td>
<td>Intel/Linux</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>94</td>
<td>Intel/Solaris</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>54</td>
<td>Intel/Linux</td>
<td>Italy (INFN)</td>
</tr>
<tr>
<td>25</td>
<td>Intel/Linux</td>
<td>New Mexico</td>
</tr>
<tr>
<td>5</td>
<td>Intel/Linux</td>
<td>Columbia U.</td>
</tr>
<tr>
<td>10</td>
<td>Sun/Solaris</td>
<td>Columbia U.</td>
</tr>
<tr>
<td>12</td>
<td>Sun/Solaris</td>
<td>Northwestern</td>
</tr>
<tr>
<td>2510</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to solve BIG problems with Branch and Bound

1. Start with a state of the art bounding technique
2. Do intelligent search and branching
   - We employ/extend “strong branching” in this non-linear context
   - Use different branching strategies depending on the relative difficulty of a node
3. Pick a very powerful computing platform
   - The Computational Grid!
4. Fit the algorithm to the platform
MW Implementation

- Fitting the B & B algorithm into the master-worker paradigm is not ground-breaking research
- We must avoid “contention” at the master

![Diagram of master-worker paradigm with tasks being sent and received](image)
All The Queueing Theory I Know

- We can reduce contention in two ways
  1. Increase the service rate
  2. Reduce the arrival rate

- A parallel depth-first oriented strategy achieves these goals.
  - Available worker is given “deepest” node by master
  - Worker examines the subtree rooted at this node in a depth-first fashion for $t$ seconds.
Parallel Depth-First Search

- Other “standard” search strategies fail completely!
  - Too much memory required at master
  - Too many nodes passed back to master
- Don’t try this at home!
  - If you don’t have a good upper bound with which to fathom, this can fail miserably!
Truth in Advertising

- The parallel depth-first search strategy is awful too!
- Define the *parallel efficiency*:

\[
\eta = \frac{\sum \text{(Time workers spend executing tasks)}}{\sum \text{(Time workers are available)}}
\]

- In our initial implementation, \( \eta = 0.41 \)
- Since there is very little synchronization required in the algorithm, this number is shockingly low!
Deducing the Problem

- We may *want* the workers to examine a subtree for $t$ seconds, but that doesn’t mean that there are $t$ seconds of work!
- A histogram of task times:
Elementary, Dear Watson

- Make sure that workers only pass back nodes that will have enough “meat”
  - Order children so that “easy” ones are first in the DFS stack
  - Allow additional time for workers to pop up the DFS stack, finishing off remaining easy nodes.
- $\eta$ improved to 0.9
(NUG30) \((n = 30)\) has been the “holy-grail” of computational QAP research for > 30 years

In 2000, Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem

Using an old idea of Knuth, we estimated the CPU time required to solve NUG30 to be 5-10 years on a fast workstation

We’d better get a pretty power computing platform, like the Computational Grid.
NUG30 is solved!

14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, 23

“MY FATHER USED $3.46 \times 10^8$ CPU SECONDS, AND ALL I GOT WAS THIS LOUSY PERMUTATION”

<table>
<thead>
<tr>
<th>Wall Clock Time:</th>
<th>6:22:04:31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. # Machines:</td>
<td>653</td>
</tr>
<tr>
<td>CPU Time:</td>
<td>$\approx$ 11 years</td>
</tr>
<tr>
<td>Nodes:</td>
<td>11,892,208,412</td>
</tr>
<tr>
<td>LAPs:</td>
<td>574,254,156,532</td>
</tr>
<tr>
<td>Parallel Efficiency:</td>
<td>92%</td>
</tr>
</tbody>
</table>
Workers

![Graph showing fluctuation in the number of workers over time from 6/9 to 6/15.](image)
KLAPS
## Even More Wasted CPU Time

<table>
<thead>
<tr>
<th></th>
<th>KRA30B</th>
<th>KRA32</th>
<th>THO30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time (Days)</td>
<td>3.79</td>
<td>12.3</td>
<td>17.2</td>
</tr>
<tr>
<td>Avg. # Machines</td>
<td>462</td>
<td>576</td>
<td>661</td>
</tr>
<tr>
<td>Max. # Machines</td>
<td>780</td>
<td>1079</td>
<td>1307</td>
</tr>
<tr>
<td>CPU Time (Years)</td>
<td>4.32</td>
<td>15.2</td>
<td>24.7</td>
</tr>
<tr>
<td>Nodes</td>
<td>5.14 × 10^9</td>
<td>16.7 × 10^9</td>
<td>34.3 × 10^9</td>
</tr>
<tr>
<td>LAPs</td>
<td>188 × 10^9</td>
<td>681 × 10^9</td>
<td>1.13 × 10^{12}</td>
</tr>
<tr>
<td>Parallel Efficiency:</td>
<td>92%</td>
<td>87%</td>
<td>89%</td>
</tr>
</tbody>
</table>
Conclusions

- The Computational Grid is a powerful computing platform
- With the Master-Worker paradigm, it is relatively easy to implement and run on the Grid.
- With a little tuning, many algorithms can be “shoehorned” to run effectively with the Master-Worker paradigm, and hence on the Grid.
  - You have to fit your algorithm to the computational platform!
- The boost in computing power that you get by making your code/algorithm Grid-enabled is worth the effort!
The End!

We want **YOU** to use the Computational Grid and **MW**.

- Papers at [http://www.optimization-online.org](http://www.optimization-online.org)
- [http://www.cs.wisc.edu/condor/mw](http://www.cs.wisc.edu/condor/mw)
- [mailto:jtl3@lehigh.edu](mailto:jtl3@lehigh.edu)