**MW: The Master-Worker Library**

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

Paradyn/Condor Week
University of Wisconsin

Madison, WI  March 14, 2005
MWCollaborators

- Greg Thain
- Wen-Han Goh
- Sanjeev Kulkarni
- Miron Livny
- Steve Wright
- Mike Yoder
- Pete Keller
- Jichuan Chang
- Alan Bailey
- Minyi Xu

- Jean-Pierre Goux
Outline

- MW Motivation
- MW Design
- MW Successes
  - Stochastic Linear Programming
  - The Quadratic Assignment Problem—Solving nug30.
- MW Future

Meet Jeff!
Jeff wants to solve large numerical optimization problems
Jeff’s Personal Condor

Jeff’s Personal Condor Pool

Flocking

Wisconsin Condor Pool

Glide-in

NCSA SuperComputer

Argonne SuperCluster

Lehigh Condor Pool
Grid-Enabling Algorithms

- Condor and “glide-in” give Jeff the infrastructure from which to build a grid (the spare CPU cycles),
- Jeff still needs a mechanism for controlling a (large) distributed algorithm on a computational grid
- No guarantee about how long a processor will be available.
- No guarantee about when new processors will become available

To make parallel algorithms dynamically adjustable and fault-tolerant, Jeff could (should?) 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
MW : A Master-Worker Grid Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- **MW** is a software package that encapsulates these abstractions
  - API : C++ abstract classes
  - User writes 10 methods
  - The **MW**ized code will transparently adapt to the dynamic and heterogeneous computing environment
- **MW** also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/{PVM, Files, Socket}
  - Single processor
MW API

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

- **MWTask**
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()

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

---

**MW and Condor!**

- Think of MW as a more dynamic and flexible DAG-Man
- It’s also more complicated to use
MW Applications

- **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
- **MWQPBB** (Linderoth) – A (simplicial) branch and bound code for solving quadratically constrained quadratic programs
- **MWAND** (Linderoth, Shen) – A nested decomposition based solver for multistage stochastic linear programming
- **MWATR** (Linderoth, Shapiro, Wright) – A trust-region-enhanced cutting plane code for linear stochastic programming and statistical verification of solution quality.
- **MWQAP** (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem
Stochastic LP—Work-Cycle Computation

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.
Show-Off!

- (with Steve Wright), Jeff aims to show off by solving “The World’s Largest Linear Program”
- Storm – A cargo flight scheduling problem (Mulvey and Ruszczyński)
- Solve an instance with 10,000,000 scenarios
- $x \in \mathbb{R}^{121}, y_s \in \mathbb{R}^{1259}$
- The deterministic equivalent is of size

$$A \in \mathbb{R}^{985,032,889 \times 12,590,000,121}$$
Jeff’s 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>
TA-DA!!!!!

**Computation Statistics**

<table>
<thead>
<tr>
<th>Description</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>

Jeff Linderoth

MW: The Master-Worker Library
Number of Workers
The Quadratic Assignment Problem

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

- Assign facilities to locations minimizing total distance flow between facilities must travel
- QAP is NP-Hard
  - Branch-and-bound is the method of choice
Tree-Based Computations

- Feasible solution $\Rightarrow$ upper bound
- Relaxed problem $\Rightarrow$ lower bound

**Branch-and-Bound**

1. Is solution to relaxed problem feasible?
   - Yes?  YAHOO!
The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research.
- We must avoid contention at the master:
  - Reduce arrival rate: Have machines work on a task for a sufficiently long time (Dynamic Grain Size).
  - Increase service rate: Do not have workers pass back many nodes. Keep master’s list of tasks small.
- Balancing efficiency considerations with search considerations was very important! (50% → 90%)
- We contend that with appropriate tuning, many algorithms can be shoehorned into the master-worker paradigm!

MW can be a grid computing workhorse!
The Holy Grail

- nug30 (a QAP instance of size 30) had been the “holy grail” of computational QAP research for $> 30$ years
- In 2000, Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem
- Using a mathematically sophisticated and well-engineered algorithm, we still estimated that we would require 11 CPU years to solve the problem.
The nug30 Computational Grid

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Location</th>
<th>How</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>SGI/Irix</td>
<td>Argonne</td>
<td>Glide-in</td>
</tr>
<tr>
<td>414</td>
<td>Intel/Linux</td>
<td>Argonne</td>
<td>Glide-in</td>
</tr>
<tr>
<td>1024</td>
<td>SGI/Irix</td>
<td>NCSA</td>
<td>Glide-in</td>
</tr>
<tr>
<td>16</td>
<td>Intel/Linux</td>
<td>NCSA</td>
<td>Flocked</td>
</tr>
<tr>
<td>45</td>
<td>SGI/Irix</td>
<td>NCSA</td>
<td>Flocked</td>
</tr>
<tr>
<td>246</td>
<td>Intel/Linux</td>
<td>Wisconsin</td>
<td>Flocked</td>
</tr>
<tr>
<td>146</td>
<td>Intel/Solaris</td>
<td>Wisconsin</td>
<td>Flocked</td>
</tr>
<tr>
<td>133</td>
<td>Sun/Solaris</td>
<td>Wisconsin</td>
<td>Flocked</td>
</tr>
<tr>
<td>190</td>
<td>Intel/Linux</td>
<td>Georgia Tech</td>
<td>Flocked</td>
</tr>
<tr>
<td>96</td>
<td>Intel/Solaris</td>
<td>Georgia Tech</td>
<td>Flocked</td>
</tr>
<tr>
<td>54</td>
<td>Intel/Linux</td>
<td>Italy (INFN)</td>
<td>Flocked</td>
</tr>
<tr>
<td>25</td>
<td>Intel/Linux</td>
<td>New Mexico</td>
<td>Flocked</td>
</tr>
<tr>
<td>12</td>
<td>Sun/Solaris</td>
<td>Northwestern</td>
<td>Flocked</td>
</tr>
<tr>
<td>5</td>
<td>Intel/Linux</td>
<td>Columbia U.</td>
<td>Flocked</td>
</tr>
<tr>
<td>10</td>
<td>Sun/Solaris</td>
<td>Columbia U.</td>
<td>Flocked</td>
</tr>
<tr>
<td>2510</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

**Computation Statistics**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time:</td>
<td>6:22:04:31</td>
</tr>
<tr>
<td>Avg. # Machines:</td>
<td>653</td>
</tr>
<tr>
<td>CPU Time:</td>
<td>≈ 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 the number of workers over time]

-Time axis from 6/9 to 6/15
-Number of workers ranging from 0 to 1800

Jeff Linderoth
MW: The Master-Worker Library
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 \times 10^9)</td>
<td>(16.7 \times 10^9)</td>
<td>(34.3 \times 10^9)</td>
</tr>
<tr>
<td>LAPs</td>
<td>(188 \times 10^9)</td>
<td>(681 \times 10^9)</td>
<td>(1.13 \times 10^{12})</td>
</tr>
<tr>
<td>Parallel Efficiency:</td>
<td>92%</td>
<td>87%</td>
<td>89%</td>
</tr>
</tbody>
</table>
MW rollout

- **MW (0.1)** available from the Condor web page
  - Web: [http://www.cs.wisc.edu/condor/mw](http://www.cs.wisc.edu/condor/mw)
- **Mailing List**
  - Email: majordomo@cs.wisc.edu with email body
    - subscribe mw
- **Bugzilla**
  - [http://coral.ie.lehigh.edu/cgi-bin/bugzilla/index.cgi](http://coral.ie.lehigh.edu/cgi-bin/bugzilla/index.cgi)
  - [mailto:jtl3@lehigh.edu](mailto:jtl3@lehigh.edu) to become registered user
MWRollout

The Good News!

- **MW** (0.2). It’s getting better and better!
  - Everyone thank Greg Thain!
- Improved robustness: Bug Fixes and Code Scrubbing
- User’s Guide
- New (better) examples: knapsack solver with branch-and-bound
- Ported to new platforms: (x86_64, cygwin)
- Part of NMI nightly build procedure

The Bad News!

**MW** 0.2 will be available “soon”
Conclusions

1. If your parallel algorithm is not “pleasantly” parallel, or requires *dynamic* configuration of tasks, then the master-worker paradigm might be right for you.

2. The master-worker paradigm is very nicely suited to a Grid implementation
   - We really believe that master-worker is the “right” paradigm for distributed computing on the Grid

3. MW can make implementing master-worker algorithms for the Grid easier
Tell Us!

We want **YOU** to tell us what you want **MW** to be

1. Easier User Interfaces (C/Python/Java)?
2. Different Communication Interfaces? (MPI?)
3. Support for worker to be “black-box” executable?
4. High-Level Language (matlab/octave), akin to GridSolve?
5. How big do you want to scale?
The End!

We want **YOU** to join the **MW** community of users

http://www.cs.wisc.edu/condor/mw
mailto:mw@cs.wisc.edu