Flexible Coscheduling

Eitan Frachtenberg\textsuperscript{1,2}, Dror Feitelson\textsuperscript{2}, Fabrizio Petrini\textsuperscript{1}, Juan Fernandez\textsuperscript{1}

\textsuperscript{1} CCS-3 Modeling, Algorithms, and Informatics Group
Computer and Computational Sciences (CCS) Division
Los Alamos National Laboratory
{eitanf,fabrizio,juanf}@lanl.gov

\textsuperscript{2} School of Computer Science and Engineering
Hebrew University, Jerusalem, Israel
feit@cs.huji.ac.il

IPDPS 2003
Outline

- Parallel job scheduling
  - Where we are
  - Recent challenges and opportunities
Outline

- Parallel job scheduling
  - Where we are
  - Recent challenges and opportunities

- Flexible coscheduling
  - New job scheduling method
  - Various kinds of applications and workloads
Outline

- Parallel job scheduling
  - Where we are
  - Recent challenges and opportunities

- Flexible coscheduling
  - New job scheduling method
  - Various kinds of applications and workloads

- Performance
  - Synthetic tests
  - Real applications
  - Dynamic workloads
Parallel Job Scheduling - Space Slicing

- Processors are divided to partitions
- Various implementations (CM-5, SP2, Cray T3D, BG/L)
- Each job runs to completion in its dedicated partition
- Batch scheduling - no preemption
Parallel Job Scheduling - Space Slicing

- Processors are divided to partitions
- Various implementations (CM-5, SP2, Cray T3D, BG/L)
- Each job runs to completion in its dedicated partition
- Batch scheduling - no preemption
Parallel Job Scheduling - Time Slicing

- Multiprogramming in a parallel machine
- Improve utilization, response time, interactivity
Parallel Job Scheduling - Time Slicing

- Multiprogramming in a parallel machine
- Improve utilization, response time, interactivity

**Challenges:**
- Scalability: machines and clusters are growing
- Overhead, cache, and memory pressure
- Flexibility: various jobs and workloads:
  - Cooperating processes need to be scheduled together
  - Load imbalance
Explicit Coscheduling

- Gang Scheduling (GS): coordinated context switching
- Context switch incurs overhead and cache pressure
- Scalability issues with global context switch
Explicit Coscheduling

- Gang Scheduling (GS): coordinated context switching
- Context switch incurs overhead and cache pressure
- Scalability issues with global context switch
Implicit Coscheduling

- Various methods: DCS, SB, PBT, ICS,...
- Use only local information for coordination
- Good for load-imbalance and utilization
- So-so for fine-grained or rate-equivalent jobs
Implicit Coscheduling

- Various methods: DCS, SB, PBT, ICS,...
- Use only local information for coordination
- Good for load-imbalance and utilization
- So-so for fine-grained or rate-equivalent jobs
Time-Slicing Scheduling

\[ \text{DCS}^{\text{[Sobalvarro98]}} \quad \text{BCS}^{\text{[Petrini00]}} \quad \text{GS}^{\text{[Ousterhout82]}} \]

- Less coordination
- Load-imbalance
- Simplicity
- Fine-grained
- Rate-equivalence

- More coordination
- Fine-Grained
- Fair
- Scalability
- Overhead
Time-Slicing Scheduling

- **SB** [Dusseau01] - less
- **DCS** [Sobalvarro98]
- **BCS** [Petrini00] - coordination
- **GS** [Ousterhout82] - more

- ✓ Load-imbalance
- ✓ Simplicity
- ✗ Fine-grained
- ✗ Rate-equivalence
- ✓ Fine-Grained
- ✓ Fair
- ✗ Scalability
- ✗ Overhead
Flexible Coscheduling (FCS)

- Use global coordination with local information
- Monitor processes’ communication activity
- Classify processes based on communication
- Schedule processes according to their needs
FCS Decision Tree

Granularity

Fine grain
Coscheduling
Effective
CS

Coarse grain
Load balancing
Ineffective
F
Requires
RE

Oblivious
DC
FCS Scheduling

Use regular time-slices, but schedule processes based on classification:

- Fine-grained (CS) use explicit coscheduling
- Coarse-grained (DC) use no coordination
  - Local UNIX scheduler
- Load-imbalanced (F) use implicit coscheduling
  - Prioritized Spin-Block
Efficient Job Scheduling with STORM

FCS fully implemented with STORM - Scalable Tool for Resource Management

- Lightweight mechanisms, using HW collective communication primitives
- Extremely scalable - “local” context-switch and job launching costs on thousand of nodes
- Set of layered, modular daemons (per node and per machine)
- “Pluggable” scheduling algorithms: Batch, Backfilling, Gang-Scheduling, Spin-block, Local, FCS, BCS
Performance Evaluation

1. Verification tests - synthetic applications based on BSP model
2. Static workloads with real applications
3. Dynamic workloads

FCS compared to GS, SB, FCFS, and Local

Run on the ’Crescendo’ cluster:

- 32 Dual Pentium-III 1-GHz, 1-GB RAM
- Quadrics Elan3 NICs and switch
Fine-Grained Jobs

- Two fine-grained jobs run concurrently on same nodes
- Each job computes & communicates every $5ms$ ($60s$ total)
- 2 nodes, 4 processors

![Diagram showing 4 processes per iteration with two jobs labeled job 0 and job 1, and run time marked on the left and right sides.]
## Fine-Grained Jobs - Turnaround Time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Job 0</th>
<th>Job 1</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>60.00</td>
<td>120.0</td>
<td>120.0</td>
</tr>
<tr>
<td>Local</td>
<td>234.8</td>
<td>231.0</td>
<td>234.8</td>
</tr>
<tr>
<td>GS</td>
<td>118.1</td>
<td>118.1</td>
<td>118.1</td>
</tr>
<tr>
<td>SB</td>
<td>125.4</td>
<td>125.4</td>
<td>125.4</td>
</tr>
<tr>
<td>FCS</td>
<td>118.3</td>
<td>118.4</td>
<td>118.4</td>
</tr>
</tbody>
</table>
Load-Imbalanced Jobs

- Same two jobs, but with load-imbalance
- Half the processes compute twice as much
- Complementing halves create opportunity for packing

Diagram:

- 4 processes
- Job 0
- Job 1
## Imbalanced Jobs - Turnaround Time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Job 0</th>
<th>Job 1</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>116.6</td>
<td>233.6</td>
<td>233.6</td>
</tr>
<tr>
<td>Local</td>
<td>301.8</td>
<td>300.8</td>
<td>301.8</td>
</tr>
<tr>
<td>GS</td>
<td>231.3</td>
<td>231.9</td>
<td>231.9</td>
</tr>
<tr>
<td>SB</td>
<td>177.9</td>
<td>179.5</td>
<td>179.5</td>
</tr>
<tr>
<td>FCS</td>
<td>176.3</td>
<td>177.6</td>
<td>177.6</td>
</tr>
</tbody>
</table>
Complementing Jobs

- Four jobs, one with load-imbalance
- Half the processes compute four times as much
- Complementing parts create opportunity for packing

Diagram:

- Four processes
- Job 0
- Job 1
- Job 2
- Job 3

Run time per iteration
## Complementing Jobs - Turnaround Time

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>231.3</td>
<td>290.2</td>
<td>349.8</td>
<td>408.6</td>
<td>408.8</td>
</tr>
<tr>
<td>Local</td>
<td>356.1</td>
<td>233.1</td>
<td>233.6</td>
<td>233.7</td>
<td>356.1</td>
</tr>
<tr>
<td>GS</td>
<td>404.7</td>
<td>232.1</td>
<td>232.2</td>
<td>232.2</td>
<td>404.7</td>
</tr>
<tr>
<td>SB</td>
<td>261.2</td>
<td>229.2</td>
<td>229.2</td>
<td>229.2</td>
<td>261.2</td>
</tr>
<tr>
<td>FCS</td>
<td>236.3</td>
<td>233.4</td>
<td>233.5</td>
<td>232.0</td>
<td>236.3</td>
</tr>
</tbody>
</table>
SWEEP3D Performance

- Particle transport code from the ASCI workload
- Balanced, fine-grained BSP application
- In this test: run time of \( \approx 48s \) with \( 3.5ms \) granularity
- Four concurrent copies on entire cluster (64 PEs)
SWEEP3D Performance

- Particle transport code from the ASCI workload
- Balanced, fine-grained BSP application
- In this test: run time of $\approx 48s$ with $3.5ms$ granularity
- Four concurrent copies on entire cluster (64 PEs)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>193.0</td>
</tr>
<tr>
<td>GS</td>
<td>194.6</td>
</tr>
<tr>
<td>SB</td>
<td>208.5</td>
</tr>
<tr>
<td>FCS</td>
<td>197.5</td>
</tr>
</tbody>
</table>
SAGE Performance

- Grid Eulerian hydro code from the ASCI workload
- Imbalanced, variable granularity
- Three concurrent copies, different input parameters
- Dedicated run times of about 39s, 86s, and 95s (≈ 220s total)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>39.2</td>
<td>125.4</td>
<td>220.2</td>
<td>220.2</td>
</tr>
<tr>
<td>GS</td>
<td>120.4</td>
<td>222.0</td>
<td>227.0</td>
<td>227.0</td>
</tr>
<tr>
<td>SB</td>
<td>124.2</td>
<td>190.0</td>
<td>200.5</td>
<td>200.5</td>
</tr>
<tr>
<td>FCS</td>
<td>112.9</td>
<td>195.0</td>
<td>205.8</td>
<td>205.8</td>
</tr>
</tbody>
</table>

Flexible Coscheduling – p.22/27
## SAGE Performance

- Grid Eulerian hydro code from the ASCI workload
- Imbalanced, variable granularity
- Three concurrent copies, different input parameters
- Dedicated run times of about 39s, 86s, and 95s (≈ 220s total)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>39.2</td>
<td>125.4</td>
<td>220.2</td>
<td>220.2</td>
</tr>
<tr>
<td>GS</td>
<td>120.4</td>
<td>222.0</td>
<td>227.0</td>
<td>227.0</td>
</tr>
<tr>
<td>SB</td>
<td>124.2</td>
<td>190.0</td>
<td>200.5</td>
<td>200.5</td>
</tr>
<tr>
<td>FCS</td>
<td>112.9</td>
<td>195.0</td>
<td>205.8</td>
<td>205.8</td>
</tr>
</tbody>
</table>
1000 jobs with dynamic job arrivals, sizes and runtimes

Based on detailed model [Lublin01]

Synthetic test application with different granularities from $5ms$ to $500ms$

Modify offered load by factoring run times

Multiprogramming level of 6

Timeslice of $50ms$
Dynamic Workload - Response Time

![Graph showing response time against offered load for different coscheduling policies: FCFS, SB, GS, FCS.](image-url)
Conclusions

- FCS designed to combine the best of both worlds: explicit and implicit coscheduling.
- Monitor processes and schedule according to needs.
- Competitive with batch, local, gang, and implicit scheduling methods in varied scenarios.
- Improved job packing and handling of load-imbalance lead to lower loads and better response times.

For more information:
http://www.cs.huji.ac.il/~etcs
email: etcs@cs.huji.ac.il
Parameter Space

Granularity (sec)

Variance

Turnaround time (sec)

Flexible Coscheduling – p.26/27
STORM Demo at SC’02