Efficient Multiprogramming for Multicores with SCAF

Timothy Creech, Aparna Kotha, Rajeev Barua
University of Maryland, College Park, MD

This work was supported by a NASA Office of the Chief Technologist’s Space Technology Research Fellowship.
Overview

- Introduction
  - Current Systems
  - Objective

- Techniques
  - Dynamic Allocation Adjustment
  - Sum-speedup Allocation Policy
  - Serial “Experiments”

- Evaluation
  - Summary
  - Example Scenario
Current Systems

- Increasingly parallel processors
- Increasingly parallel software
- Multiprogramming parallel processes is difficult
  - Current operating systems schedule threads, not processes
  - The problem of managing parallelism is largely left to the user

- Multiprogramming: running multiple processes simultaneously
Current Systems

- Consider a single parallel application “A” running alone
- The OS schedules 8 threads to 8 cores – excellent!
Current Systems

- What if a second parallel process exists?
- The OS has to share 8 cores among 16 busy threads
- The machine is now oversubscribed
Current Systems

- What’s the problem with oversubscription?
  - Unnecessarily reduces granularity
  - Increased overhead by context switching
  - Hardware contention
  - Certain synchronization constructs assume no oversubscription
Current Systems

What can we do?
Current Systems

- What can we do?
- Make the thread scheduler’s job easier
  - Avoid oversubscription with **space sharing**
Current Systems

- What can we do?
- Make the thread scheduler’s job easier
  - Avoid oversubscription with **space sharing**

```
OMP_NUM_THREADS=4
```

![Diagram showing hardware contexts](image-url)
Current Systems

- Example: NAS SP+LU on 8-core Xeon E5410
  - Unmodified system: oversubscription
  - Equipartitioned space sharing: Each process gets half of the machine

<table>
<thead>
<tr>
<th>System</th>
<th>Threads</th>
<th>SP Speedup</th>
<th>LU Speedup</th>
<th>Σ Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmodified</td>
<td>8+8=16</td>
<td>1.18</td>
<td>1.78</td>
<td>2.96</td>
</tr>
<tr>
<td>Equi-partitioning</td>
<td>4+4=8</td>
<td>1.33</td>
<td>2.84</td>
<td>4.17</td>
</tr>
</tbody>
</table>

- System 40% more efficient with equipartitioning
Current Systems

- This is a lot to ask of users
  - Must be aware of the parallel runtime
  - Must manually coordinate processes
- More difficult than serial multiprogramming

Hardware Contexts

```
OMP_NUM_THREADS=4
```

```
OMP_NUM_THREADS=4
```
Current Systems

- Equipartitioning is rare in practice
  - Exists in literature, no popular solution
  - Difficult to do manually as a user

Hardware Contexts

OMP_NUM_THREADS=4

[Diagram of hardware contexts with nodes A and B and threads C0-C7]
Current Systems

- Now, say “A” gets good speedups
- But “B” has limited parallelism, speeds up significantly less

OMP_NUM_THREADS=4

Hardware Contexts

C0 C1 C2 C3 C4 C5 C6 C7
Current Systems

- Allow A to use more threads:
  - make better use of hardware contexts
  - Sum of speedups is improved

```
OMP_NUM_THREADS=6
```

```
OMP_NUM_THREADS=20
```

```
OMP_NUM_THREADS=60
```

Speedup

Threads

Hardware Contexts

A

B

C0  C1  C2  C3  C4  C5  C6  C7
### Current Systems

- **Example:** NAS SP+LU on 8-core Xeon E5410
  - Unmodified system: oversubscription
  - Equipartitioned space sharing: Each process gets half of the machine

<table>
<thead>
<tr>
<th>System</th>
<th>Threads</th>
<th>SP Speedup</th>
<th>LU Speedup</th>
<th>Σ Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmodified</td>
<td>8+8=16</td>
<td>1.18</td>
<td>1.78</td>
<td>2.96</td>
</tr>
<tr>
<td>Equi-partitioning</td>
<td>4+4=8</td>
<td>1.33</td>
<td>2.84</td>
<td>4.17</td>
</tr>
<tr>
<td>“Smart” partitioning</td>
<td>2+6=8</td>
<td>1.26</td>
<td>4.81</td>
<td>6.07</td>
</tr>
</tbody>
</table>

- System another **45%** more efficient with intelligent partitioning
Objective

“SCAF” (Scheduling and Allocation with Feedback)

1. Improve system efficiency
   - Attempt to maximize the sum of speedups
   - No oversubscription: dynamic space sharing
   - Allocate based on run-time feedback
   - Achieve higher efficiency than equipartitioning
Objective

“SCAF”

(Scheduling and Allocation with Feedback)

2. Convenience for users
   - Automatic partitioning of contexts
   - No offline profiling
   - No porting, modification, or recompilation of binaries
   - As simple as serial multiprogramming
Techniques

- Allocation Adjustment
- Sum-speedup Allocation Policy
- Serial “Experiments”
Dynamic Allocation Adjustment

- OpenMP code generation example:

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```
Dynamic Allocation Adjustment

- OpenMP code generation example:
- Parallel runtime (GOMP) spawns threads according to hardware

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```
OpenMP code generation example:

- Program also spends some time in serial sections between parallel sections

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```
Dynamic Allocation Adjustment

- OpenMP code generation example:
  - Spawn/join parallelism
  - Note: serial/parallel phases always alternate

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```

![Diagram showing serial, parallel, and serial phases with time points t0, t1, t2, t3, t4, t5, t6, t7, and t0 again, indicating the dynamic allocation adjustment process.]
The parallel loop is **malleable**

Can run on *any number* of threads

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```
Dynamic Allocation Adjustment

- Modify OpenMP runtime library
- Control parallelism of **malleable** loops

```c
unsigned time, I;
for(time=0; time<TMAX; time++){
    #pragma omp parallel for
    for(i=0; i<IMAX; i++)
        process(i, data);
    aggregate(data);
}
```
Dynamic Allocation Adjustment

Instead of users manually (and statically) space sharing...

Hardware Contexts

OMP_NUM_THREADS=4

A

C0 C1 C2 C3

B

C4 C5 C6 C7

OMP_NUM_THREADS=4
Dynamic Allocation Adjustment

- Instead of users manually and statically space sharing
- Processes use a SCAF port of the OpenMP runtime

(Blueprint indicates loaded SCAF OpenMP shared object)
Dynamic Allocation Adjustment

- The runtime consults a SCAF daemon
  - scafd is system-wide and centralized
  - Userspace only – no kernel modifications

Diagram:
- scafd
- Hardware Contexts (C0 to C7)
  - A
  - B

Network connections:
- A to B
- C0 to C3
- C4 to C7
Dynamic Allocation Adjustment

- **scafd** can make allocation decisions dynamically
- Allocation decisions respected at each process’s next parallel spawn
Dynamic Allocation Adjustment

- Note: processes A and B are **unaware** of SCAF
- Only the compiler-dependent parallel runtime communicates with the SCAF daemon
- Enable SCAF runtimes with OS’s dynamic linker/loader
Sum-speedup Allocation Policy

- Program content and completion time are unknown
- The daemon (scafd) can only reason about instantaneous performance, e.g., IPC, Flop/s
- Most general work metric: speedup over serial execution
- Make partitioning decisions which attempt to maximize the sum of speedups

Each parallel process reports efficiency metric to scafd

\[
\text{efficiency} = \frac{\text{speedup}}{\text{threads}}
\]
Scafd uses this single efficiency metric (E) to fit each process to a simple sub-linear speedup function \( S(p) \):

\[
S(p) \approx 1 + C \cdot \log(p), \quad \text{where} \quad C \leftarrow \frac{E \cdot \log(p') - 1}{\log(p')}
\]

Log-speedup chosen for single coefficient and diminishing returns
This has an intuitive closed solution:

\[ p_i \leftarrow \frac{N \cdot C_i}{\sum_j C_j} \]

where

- \( N \) is the number of processes
- \( C_k \) is the fitting coefficient of process \( k \)
- \( p_k \) is the allocation of processes \( k \)

In other words, each process receives an allocation proportional to its fitting coefficient.
scafd uses this policy to advise processes on allocations

Sum-speedup Allocation Policy
Processes report efficiencies after parallel sections

<table>
<thead>
<tr>
<th>PID</th>
<th>E</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>0.7</td>
<td>2</td>
</tr>
</tbody>
</table>
Scafd immediately responds with pre-computed allocations.

<table>
<thead>
<tr>
<th>PID</th>
<th>E</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>0.7</td>
<td>2</td>
</tr>
</tbody>
</table>

Hardware Contexts:

- C0
- C1
- C2
- C3
- C4
- C5
- C6
- C7
**Sum-speedup Allocation Policy**

- scafd re-evaluates allocations periodically (~4Hz)

### Hardware Contexts
- A
- B
- C0
- C1
- C2
- C3
- C4
- C5
- C6
- C7

### Allocation Table

<table>
<thead>
<tr>
<th>PID</th>
<th>E</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>0.7</td>
<td>3</td>
</tr>
</tbody>
</table>
Sum-speedup Allocation Policy

...and communicates allocation changes

**PID** | **E** | **Threads**
---|---|---
A | 0.5 | 5
B | 0.7 | 3

Hardware Contexts

![Diagram showing allocation policy with nodes A and B connected by dashed lines and hardware contexts C0 to C7 with numbers 0 to 7. The nodes A and B are connected to hardware contexts C0 to C7 through arrows indicating the allocation policy.]
Sum-speedup Allocation Policy

(Lower efficiency can receive higher allocation if achieved on many threads)

<table>
<thead>
<tr>
<th>PID</th>
<th>E</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>0.7</td>
<td>3</td>
</tr>
</tbody>
</table>

Hardware Contexts:

- A
- B
- C0
- C1
- C2
- C3
- C4
- C5
- C6
- C7

`scafd` threads=5

- A threads=5
- B threads=3
Serial Experiments

Problem: how do processes compute their efficiency?

- Missing information: serial performance
- Don’t want off-line profiling
- Can’t wait to temporarily serialize

64-core processor
Serial Experiments

- Solution: Serial “Experiments”
  - 1 context used for experiment
  - Child process (not thread) executes the section serially
  - OS tracing facilities used to protect parent process correctness
  - Performance measured as hardware counter rates
  - Experiment ends early if:
    - Parallel section finishes
    - Correctness is in danger
Serial Experiments

- Solution: Serial “Experiments”
  - Process uses 1 context to measure serial performance
  - Results used later to compute speedup/efficiency

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

Unrolled execution of example loop

64-core processor
Serial Experiments

Example of serial experiment in process A:

First time the parallel section is seen: use 1 core to run a serialized version

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

The remaining hardware contexts work on progressing through the section

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

Performance results are recorded from the experiment and parallel section

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

Parallel section’s efficiency is estimated as
\[
\frac{2.4/0.6}{8-1} = 0.57
\]
Serial Experiments

Example of serial experiment in process A:

Serial phases proceed normally, with speedup of 1, efficiency $1/8 = 0.125$
Serial Experiments

Example of serial experiment in process A:

Same parallel section: no need to re-run the serial experiment

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

The latest achieved parallel performance is recorded

Unrolled execution of example loop

E: 0.12 0.57 0.12 0.12

2.6 IPC
Serial Experiments

Example of serial experiment in process A:

Latest efficiency for this section estimated at \((2.6/0.6) / 8 = 0.54\)

Unrolled execution of example loop
Serial Experiments

Example of serial experiment in process A:

And so on!

Unrolled execution of example loop

<table>
<thead>
<tr>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
<td>0.54</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
</tr>
</tbody>
</table>

E: 0.12 0.57 0.12 0.12 0.54 0.12 0.12 0.12 0.55 0.12
Serial Experiments

Example of serial experiment in process A:

Efficiencies are passed through a low-pass filter, accounting for durations. This “smooths” the alternating serial/parallel efficiencies present in applications.

E: 0.12 0.57 0.12 0.12 0.54 0.12 0.12 0.52 0.12 0.12 0.55 0.12

Unrolled execution of example loop
Efficiencies are passed through a low-pass filter, accounting for durations. This “smoothes” the alternating serial/parallel efficiencies present in applications.

**Low-pass filter:** attenuates high-frequency signal content
Evaluation

- Summary
- Example Scenario
Summary

- Pairwise (2-way) multiprogramming of NAS benchmarks
- 57% (Sparc T2) and 70% (Xeon) of benchmarks see improvement over both equipartitioning and oversubscription
- Mean improvement was 15% increase in sum of both speedups over fastest
Summary

- Pairwise (2-way) multiprogramming of NAS benchmarks
  - 57% (Sparc T2) and 70% (Xeon) of benchmarks see improvement over both equipartitioning and oversubscription
  - Mean improvement was 15% increase in sum of both speedups over fastest
  - Up to 57% (T2) and 307% (Xeon) improvement over oversubscription
Summary

Pairwise NAS results on Sparc T2

- Benchmark Pair
- Unmodified
- Equipartitioning
- SCAF

<table>
<thead>
<tr>
<th>Benchmark Pair</th>
<th>Sum Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ua.A+mg.C</td>
<td></td>
</tr>
<tr>
<td>ua.A+lu.B</td>
<td></td>
</tr>
<tr>
<td>ua.A+ft.B</td>
<td></td>
</tr>
<tr>
<td>ua.A+bt.B</td>
<td></td>
</tr>
<tr>
<td>sp.B+ua.A</td>
<td></td>
</tr>
<tr>
<td>sp.B+mg.C</td>
<td></td>
</tr>
<tr>
<td>sp.B+lu.B</td>
<td></td>
</tr>
<tr>
<td>sp.B+ft.B</td>
<td></td>
</tr>
<tr>
<td>sp.B+bt.B</td>
<td></td>
</tr>
<tr>
<td>mg.C+lu.B</td>
<td></td>
</tr>
<tr>
<td>mg.C+bt.B</td>
<td></td>
</tr>
<tr>
<td>ft.B+mg.C</td>
<td></td>
</tr>
<tr>
<td>ft.B+lu.B</td>
<td></td>
</tr>
<tr>
<td>ft.B+bt.B</td>
<td></td>
</tr>
<tr>
<td>cg.C+ua.A</td>
<td></td>
</tr>
<tr>
<td>cg.C+sp.B</td>
<td></td>
</tr>
<tr>
<td>cg.C+mg.C</td>
<td></td>
</tr>
<tr>
<td>cg.C+lu.B</td>
<td></td>
</tr>
<tr>
<td>cg.C+ft.B</td>
<td></td>
</tr>
<tr>
<td>cg.C+bt.B</td>
<td></td>
</tr>
<tr>
<td>bt.B+lu.B</td>
<td></td>
</tr>
</tbody>
</table>
Summary

Pairwise NAS results on 2x Xeon E5410

- Benchmark Pair
- Unmodified
- Equipartitioning
- SCAF

<table>
<thead>
<tr>
<th>Benchmark Pair</th>
<th>Unmodified</th>
<th>Equipartitioning</th>
<th>SCAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ua.B+mg.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ua.B+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ua.B+ft.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ua.B+bt.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp.B+ua.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp.B+mg.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp.B+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp.B+ft.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp.B+bt.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mg.C+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mg.C+bt.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mg.C+ft.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ft.C+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ft.C+bt.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+ua.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+sp.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+mg.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+ft.C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg.C+bt.B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bt.B+lu.B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Scenario

- Detail of a 3-way multiprogramming scenario:

![Graph showing system allocation over time]
Example Scenario

- Detail of a 3-way multiprogramming scenario:
  - In this case, **45%** improvement vs. equipartitioning

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Process</th>
<th>Runtime</th>
<th>Speedup</th>
<th>Σ Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmodified</td>
<td>CG</td>
<td>436s</td>
<td>13.1</td>
<td>31.5</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>475s</td>
<td>9.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LU</td>
<td>507s</td>
<td>8.8</td>
<td></td>
</tr>
<tr>
<td>Equi-partitioning</td>
<td>CG</td>
<td>374s</td>
<td>15.5</td>
<td>40.7</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>381s</td>
<td>12.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LU</td>
<td>350s</td>
<td>13.0</td>
<td></td>
</tr>
<tr>
<td>SCAF</td>
<td>CG</td>
<td>172s</td>
<td>35.7</td>
<td>59.3</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>374s</td>
<td>12.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LU</td>
<td>424s</td>
<td>11.1</td>
<td></td>
</tr>
</tbody>
</table>
Questions?