Preliminary Evaluation of Dynamic Load Balancing Using Loop Re-partitioning on Omni/SCASH

Yoshiaki Sakae*1, Satoshi Matsuoka*1 *2, Mitsuhsisa Sato*3 and Hiroshi Harada*4

*1 Tokyo Institute of Technology, Japan
*2 JST, Japan
*3 Tsukuba University, Japan
*4 Compaq Computer, Japan

2003/5/13
Background

- Commodity cluster tends to be heterogeneous in performance
  - Incremental extension of nodes
  - Incremental upgrade of nodes
  - Cluster of clusters

- When a program is executed on hetero-cluster, its total performance is often dominated by the slowest host.

*We’ll abbreviate performance heterogeneous cluster as hetero-cluster*
An Example of Performance Degradation on Hetero-Cluster

Execution Time of SPLASH II Water on Hetero-Cluster

The slower nodes dominate the entire performance as if it were all 300MHz nodes.

We need a load balancing mechanism.
In This Work

- We extended Omni/SCASH to support hetero-clusters
  - Loop re-partitioning mechanism to achieve dynamic load balancing based on runtime performance
  - Page migration mechanism based on page reference counting (not yet implemented completely)
- We report the effect of loop re-partitioning on hetero-cluster
Omni/SCASH [Sato et al. '00] (http://www.pccluster.org/)

- One of the OpenMP implementation on Software DSM, SCASH [Harada et al. '98]
- Translates C or F77 + OpenMP programs into C with runtime library calls
  - Intermediate code (Xobject) is a kind of AST
    - Omni provides Java class libraries to process the AST easily
    - Each node of the AST is a Java object
  - Omni encapsulates each parallel region into a separate function which is invoked from master thread
  - A Global variable is allocated by the SCASH function and transposed to the pointer to that
Target Problems

- Load imbalance caused by runtime settings
  - Esp. when an application is executed on hetero-cluster

- Static techniques are inadequate, because the performance ratio varies on each cluster setting

- Dynamic scheduling based on runtime performance + page migration
OpenMP Schedulings

Processors = 3

- schedule(static)
- schedule(static, chunk_size)
- schedule(dynamic, chunk_size)
- schedule(guided, chunk_size)
Our Proposal: **Profiled** Scheduling

- Load balancing based on runtime self-profiling
- **Target:** parallel loops specified with the "#pragma omp for" directive
- **Measures** the execution time of the target loop on each thread
- Adjusts chunk size of the parallel loop dynamically based on measured performance

**Assumptions:**
- The application has no load imbalance inherently
- The target loop has no changes of a work load among the iterations
The Syntax of Profiled Scheduling

```cpp
#pragma omp [parallel] for
      schedule(profiled[, chunk_size[, eval_size[, eval_skip]]])
```

eval_skip specifies the initial iteration size which is executed normally
  - When eval_skip is omitted or 0, start profiling from the head of iters

eval_size specifies the size of profiling iterations
  - When eval_size is omitted, evaluation loop size: 1

chunk_size specifies the size of chunk
  - When the chunk_size is omitted or 0, divide remaining iters in a block manner based on performance ratio
  - When the chunk_size = n (n > 1), divide remaining iters cyclically based on performance ratio
Examples of Profiled

```cpp
#pragma omp [parallel] for
  schedule(profiled[, chunk_size[, eval_size[, eval_skip]]])
```

Example: Num Proc = 3, Performance Ratio = 3:1:2

- `schedule(profiled)`: Processors 0, 1, 2
- `schedule(profiled, 2, 3)`: Processors 0, 1, 2
- `schedule(profiled, 2, 3, 1)`: Processors 0, 1, 2

- : executed on proc 0
- : executed on proc 1
- : executed on proc 2
- : profiling loop
Code Translation when Profiled Scheduling is Specified

#pragma parallel omp for schedule(profiled, 10)
for (i = 0; i < n; i++) {
  LOOP_BODY;
}

static void __ompc_func(void **__ompc_args) {
  int i, lb, ub, step;
  long long start = 0, stop = 0;
  lb = 0, ub = N, step = 1;
  _ompc_profiled_sched_init(lb, ub, step, 10);
  while (_ompc_profiled_sched_next(&lb, &ub, start, stop)) {
    _ompc_profiled_get_time(&start);
    for (i = lb, i < ub; i += step) {
      LOOP_BODY;
    }
    _ompc_profiled_get_time(&stop);
  }
}
Overview of Loop Re-partitioning Algorithm

_def_profiled_sched_next()_

LR enabled? Yes
- Calc exec. speed
- Exc. speed data
- Predict exec. speed when LR is performed
- Perform LR
- Store division ratio in chunk_vect
- Assign iters based on chunk_vect

No
- Perf gain? Yes
- Perform LR
- Store division ratio in chunk_vect
- Assign iters based on chunk_vect

No
- Disable LR

Assign iters based on chunk_vect

LR: loop re-partitioning
Dynamic/Guided v.s. Profiled

■ Dynamic/Guided scheduling
  ■ Needs atomic access to the index managed centrally at every sub-loop index calculation
  ■ Involves communication on the distributed memory environment

■ Profiled scheduling
  ■ Doesn’t need the index managed centrally
  ■ Each thread has chunk size for all threads in chunk_vector
  ■ Communication occurs only after evaluation loop
    ■ When the target loop has no changes of a work load among the iterations, loop re-partitioning may complete on its first attempt
Dynamic Page Migration Idea (1/2)

- Counts the number of page faults at the SDSM level (c.f. precise page reference counting with hardware support [Nikolopoulos et al. ‘00])

- Migrates the page to the node with the most number of remote references to the given page
  - Because we can’t count local accesses directly, unnecessary page migration may occur
Dynamic Page Migration Idea (2/2)

- Keeps migration records to avoid the page ping-pong, and restore locality within several repetition
- Performs page migration only during the target parallel loops
  - Excludes unnecessary page reference data
  - Enables timely page migration based on appropriate page reference data
- We plan
  - Speculative page migration based on feedback from loop re-partitioning
  - Re-enable loop re-partitioning after page migration
Coordinate Profiled Scheduling with Page Migration

- Profiled scheduling and page migration affect each other
  - Loop re-partitioning will cause poor data locality
  - Page migration will affect performance prediction

- Exploits both profiled scheduling and page migration gradually
  - Both will reach the stable state in early stage of iterations
  - Needs some heuristics to balance both
Preliminary Evaluation

**Evaluation points**
- Overhead of profiled scheduling itself on performance homogeneous settings
- Comparison against static, dynamic and guided scheduling on performance heterogeneous environment

**Benchmark programs**
- **NPB2.3 EP (C + OpenMP version made by RWCP)**
  - Due to few communications, we can evaluate pure efficiency of profiled scheduling
- **NPB2.3 CG (C + OpenMP version made by RWCP)**
  - Data locality has large impact on performance, because there are many accesses to shared arrays
  - We can anticipate that there may be performance drops without some page migration mechanism
Evaluation Environment

- Performance heterogeneous cluster
  - Pentium III 500MHz node x 6
  - Celeron 300MHz node x 1
  - Other settings are the same
    - Intel 440BX Chipset
    - 512MB Memory
    - Myrinet M2M-PCI32C
- RedHat 7.2 (Linux-2.4.18)
- SCore-5.0.1
- gcc-2.96 -O
EP Class S Performance (Homogeneous Settings)

Execution Time of EP Class S on Homo-Cluster (chunk_size: none)

There is no overhead with profiled
EP Class S Performance

Execution Time of EP Class S on Hetero-Cluster (chunk_size: none)
CG Class A Performance

Execution Time of CG Class A on Hetero-Cluster (chunk_size: none)

- static (Cel x 1 + Pen III)
- guided (Cel x 1 + Pen III)
- profiled (Cel x 1 + Pen III)

Performance drops due to the overhead of profiled
## Breakdown of CG Class A

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Profiled</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 miss ratio</td>
<td>29.6%</td>
<td>31.1%</td>
</tr>
<tr>
<td>Page Fault at SCASH Level</td>
<td>16456</td>
<td>27201</td>
</tr>
<tr>
<td>Barrier</td>
<td>5088</td>
<td>8006</td>
</tr>
</tbody>
</table>

- More page faults with profiled, because the data access range may change on each iteration.
- More barriers with profiled, because it will repeat unnecessary profiling loops (see next figure).
Overview of Loop Re-partitioning Algorithm (again)

```c
_ompc_profiled_sched_next()
```

1. **LR enabled?**
   - Yes: Calc exec. speed
     - Yes: Perform LR
       - Store division ratio in `chunk_vect`
       - Assign iters based on `chunk_vect`
     - No: Disable LR
   - No: Barrier

2. **Perf gain?**
   - Yes: Perform LR
   - No: Assign iters based on `chunk_vect`

LR: loop re-partitioning
Conclusion

- We extended Omni/SCASH to support profiled scheduling for dynamic load balancing
- We made sure that profiled scheduling is more effective than static/dynamic/guided one on hetero-cluster with EP which are not influenced by data placement
- Profiled scheduling reveals its overhead due to changes in data access ranges
- We showed the plan of page migration extension to SCASH
Future Work

- Complete the implementation of page migration
- Integrate loop re-partitioning with page migration
- Evaluate this system with more applications