

# Towards Dynamic Load Balancing Using Page Migration and Loop Re-Partitioning on Omni/SCASH

Yoshiaki Sakae<sup>\*1</sup>, Satoshi Matsuoka<sup>\*1 \*2</sup>,  
Mitsuhisa Sato<sup>\*3</sup> and Hiroshi Harada<sup>\*4</sup>

<sup>\*1</sup> Tokyo Institute of Technology, Japan

<sup>\*2</sup> JST, Japan

<sup>\*3</sup> Tsukuba University, Japan

<sup>\*4</sup> Compaq Computer, Japan

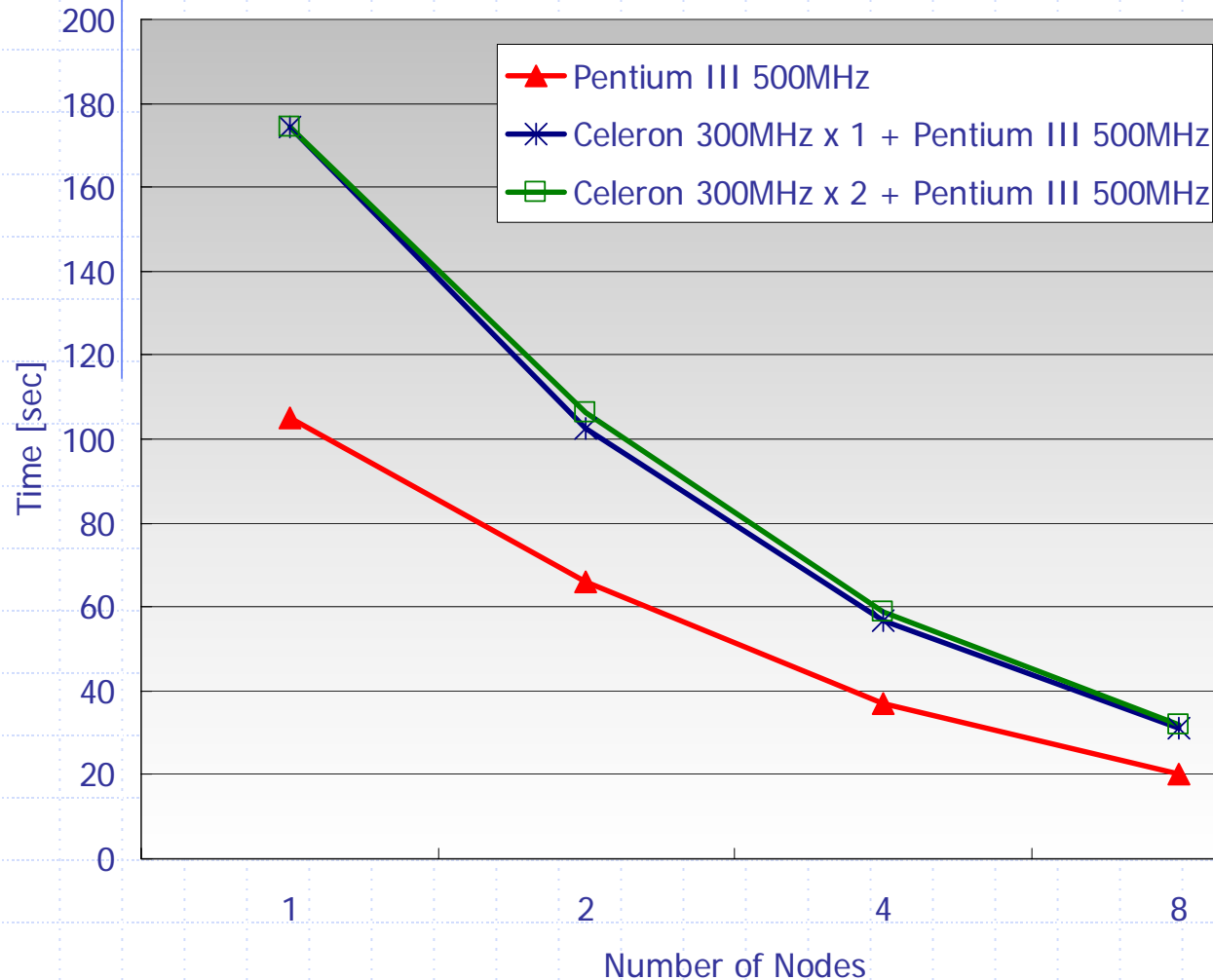
# Background

- ◆ 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, it's total performance is often **dominated by the slowest host**.

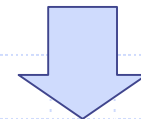
\*We'll abbreviate performance heterogeneous cluster as **hetero-cluster**

# 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 certain 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<sup>5</sup>

# Several Causes of Load Imbalance and it's Solution (1/2)

- ◆ The application has load imbalance inherently
  - ⇒ Dynamic/Guided scheduling of OpenMP + Page migration
- ◆ The load imbalance is caused by data placement
  - ⇒ Performs explicit data placement by some means or use affinity scheduling supported by Omni/SCASH

# Several Causes of Load Imbalance and it's Solution (2/2)

- ◆ When an application is executed on hetero-cluster

  - ⇒ Dynamic scheduling based on runtime performance + Page migration

- ◆ There are differences in loads among the nodes due to multi-user environment

  - ⇒ Dynamic scheduling based on runtime performance + Page migration

# Load Balancing Based on Runtime Self-Profiling

- ◆ Target: parallel loops specified with the "#pragma omp for" directive
- ◆ Loop re-partitioning based on runtime performance profiling: **profiled scheduling**
- ◆ Page migration based on runtime page reference counting
  - Loop re-partitioning may cause changes in a data access range of each thread and lead to the decrease of data locality

# Profiled Scheduling

- ◆ Measures the execution time of the target loop on each thread
- ◆ Adjusts chunk size of the parallel loop dynamically based on performance
- ◆ Assumptions:
  - The application has no load imbalance inherently
  - The target loop has no changes of a work load among the iterations

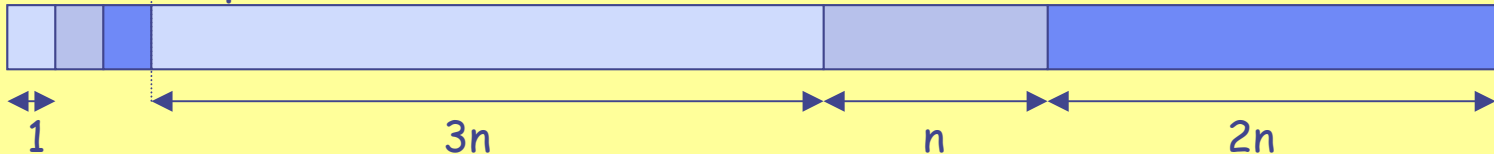
# The Format of Profiled Scheduling

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

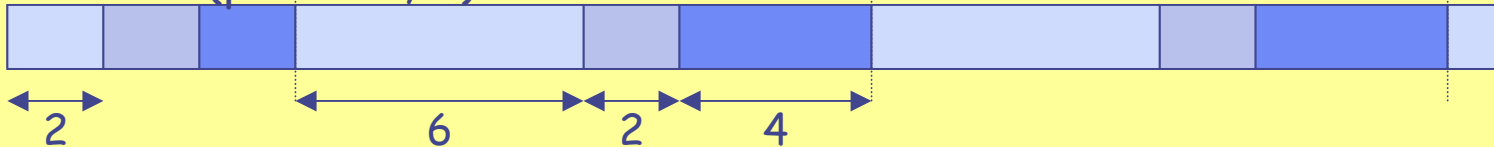
- ◆ When the `chunk_size` is omitted:
  - Evaluation loop size: 1
  - Block division mode
- ◆ When the `chunk_size` is specified:
  - Evaluation loop size: `chunk_size`
  - Cyclic division mode

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

`schedule(profiled)`



`schedule(profiled, 2)`



# Code Translation when Profiled Scheduling is Specified

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

Translated at the level of the intermediate representation of Omni

- ◆ `__ompc_func()` is invoked on the slave threads in parallel
- ◆ `_ompc_profiled_get_time()` measures the runtime performance
- ◆ `_ompc_profiled_sched_next()` calculates the next chunk of iteration based on measured performance

```
static void __ompc_func(void **__ompc_args) {
    int i, lb, ub, step;
    double start = 0.0, stop = 0.0;
    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);
    }
}
```

# The Algorithmic Overview of \_ompc\_profiled\_sched\_next() (1/2)

- A) When a loop re-partitioning is performed:
1. Calculates execution speed and broadcast it
  2. Estimates the time taken to perform the remaining iterations when performing the loop re-partitioning, and also when not performing
  3. When the performance improvement is above a certain threshold:
    - a. Each thread calculates the number of chunks for all threads and store these in the **chunk\_vector**
    - b. Each thread adjusts its own loop index, loop bounds, etc., and exit

# The Algorithmic Overview of \_ompc\_profiled\_sched\_next() (2/2)

4. When the performance improvement is not above a certain threshold:
  - a. A flag is set to indicate that loop re-partitioning may not be performed anymore.
  - b. Each thread adjusts its own loop related information based on **chunk\_vector** calculated previously, and exit

## B) When a loop re-partitioning is not performed:

1. Each thread adjusts its own loop related information based on **chunk\_vector** calculated previously, and exit

# 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 makes the page migration converge within several times
- ◆ 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

# Preliminary Evaluation

- ◆ Evaluates the effect of profiled scheduling compared to static/dynamic one
  - $\pi$  calculation
  - NPB2.3 EP (C + OpenMP version made by RWCP)

```
step = 1.0 / num_steps;
start_time = second();
#pragma omp parallel
{
    num_threads = omp_get_num_threads();
#pragma omp for reduction(+: sum) private(x) schedule(profiled, 1000)
    for (i = 0; i <= num_steps; i++) {
        x = (i - 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
}
pi = step * sum;
run_time = second() - start_time;
```

# Evaluation Environment

## ◆ Performance heterogeneous cluster

- Pentium III 500MHz node x 6
- Celeron 300MHz node x 2
- Other settings are the same
  - ◆ Intel 440BX Chipset
  - ◆ 512MB Memory
  - ◆ Myrinet M2M-PCI32C

## ◆ RedHat 7.2 (linux-2.4.18)

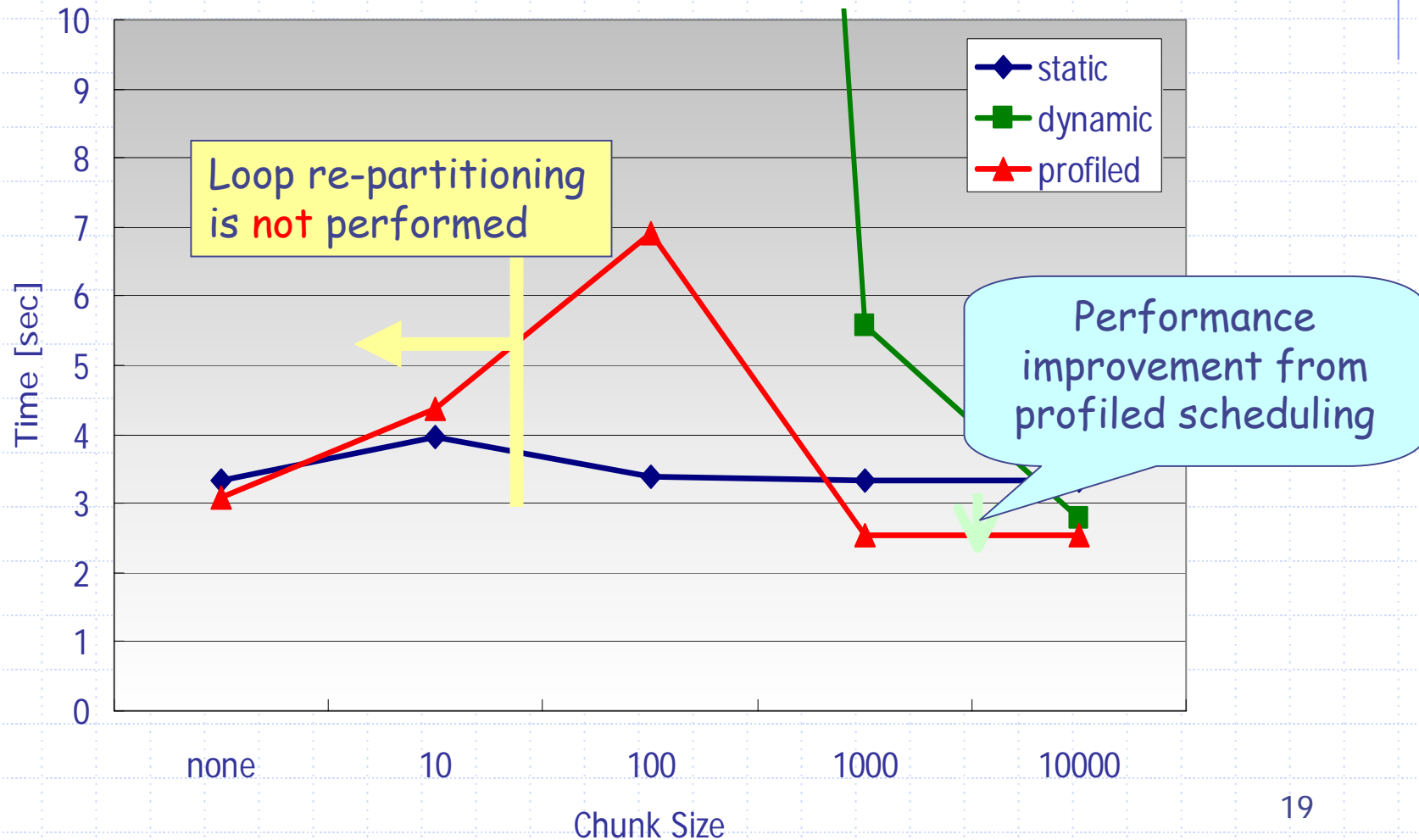
## ◆ SCore-5.0.1

## ◆ gcc-2.96 -O



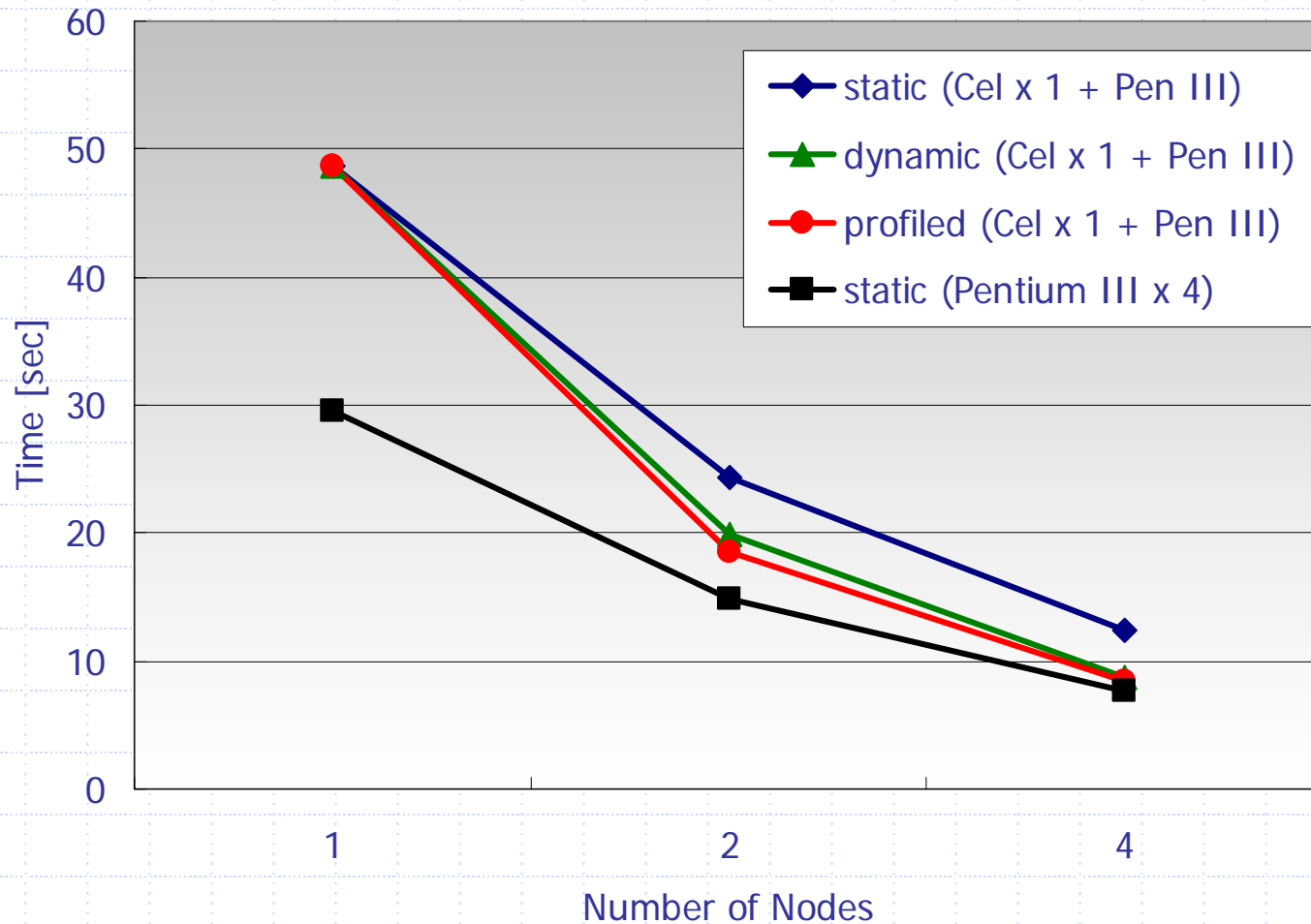
# Execution Time of $\pi$ 100M steps

PI (100M steps) on Pentium III 500MHz x 3 + Celeron 300MHz



# EP Class S Performance

Execution Time of EP Class S on Hetero-Cluster (chunk\_size: none)



# Conclusion

- ◆ We extended Omni/SCASH to support profiled scheduling for dynamic load balancing
- ◆ We showed the plan of page migration extension to SCASH
- ◆ We made sure that profiled scheduling is more effective than static/dynamic one on hetero-cluster by simple programs which are not influenced by data placement

# Future Work

- ◆ Complete the implementation of page migration
- ◆ Integrate loop re-partitioning with page migration
- ◆ Brush up thresholds used in decision of loop re-partitioning
- ◆ Evaluate this system with real applications

# EP Class S Chunk Vector on 4 nodes

- ◆ Remaining iterations after initial evaluation loops:  $252 (= 256 - 4)$
- ◆ Chunk vector: [70 70 70 43]



Celeron node