An Efficient Implementation of Data-Parallel Skeletons on Multicore Processors

Kiminori Matsuzaki
University of Tokyo

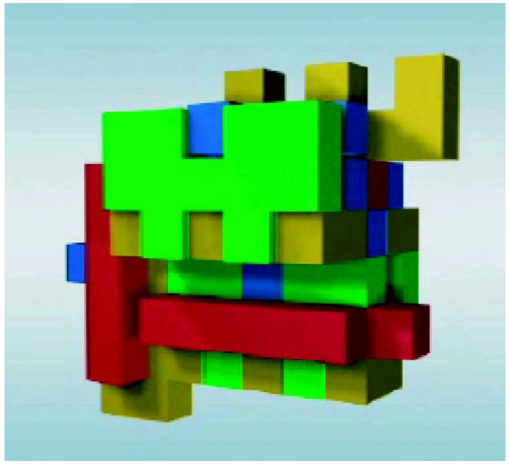
Joint work with Kazutoshi Kariya, Zhenjiang Hu, Masato Takeichi

Skeletal Parallelism [Cole 89]

- Parallel Skeletons (Algorithmic Skeletons)
  - Well-known patterns in parallel computation
  - *Sequential* interface & *Parallel* implementation

Parallel Program
= Parallel Patterns + (Sequential) Details
<Skeletons>

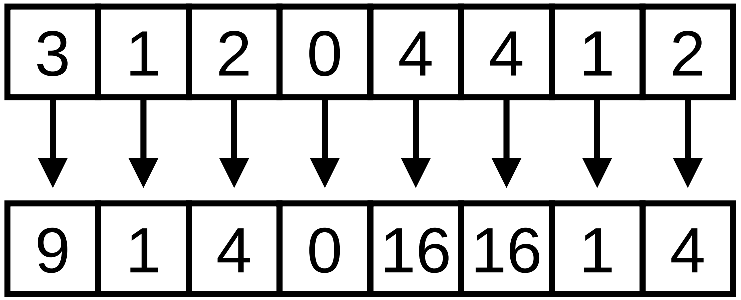
- Several merits
  - Productivity
  - Efficiency
  - Portability



Two Important Skeletons

Map:

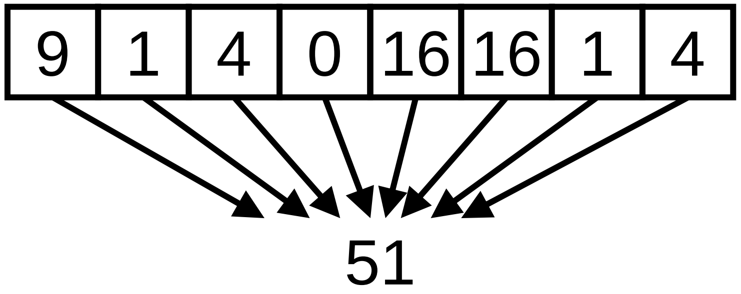
- Apply function to each element




```
for(i=0; i<n; ++i){
  b[i] = a[i]*a[i];
}
```

Reduce:

- Reduction by an associative operator




```
s = 0;
for(i=0; i<n; ++i){
  s += a[i];
}
```

Example: Computing Variance

$$var = \frac{1}{n} \sum (a_i - ave)^2 \quad \textbf{where } ave = \frac{1}{n} \sum a_i$$

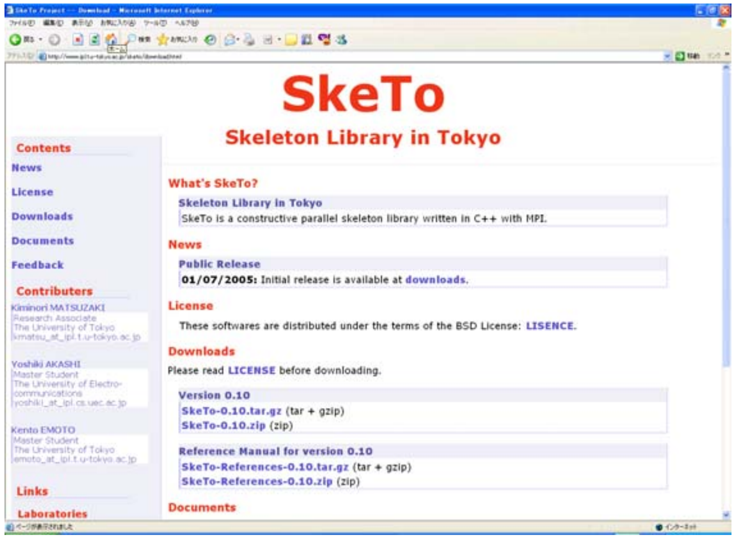
- An skeletal program

```
Var(array, n) {
   ave = reduce(+, array) / n;
   array' = map(–ave, array);
   array'' = map(^2, array');
   return reduce(+, array'') / n;
}
```

We need not consider parallelism: No send&recv!

SkeTo (*Ske*leton Library in *To*kyo)
助っ人 (Supporter or relief)

- A parallel skeleton library for non-specialist
  - Implemented in C++ & MPI
  - For distributed-memory parallel computers
  - Support for (1D or 2D) arrays, trees
  - Fusion transformation optimizer



Available online from
http://www.ipl.t.u-tokyo.ac.jp/sketo/
(ver 1.0 will be available soon)

Topics in This Talk

- Question: Is the implementation also efficient on multicore CPUs?
- Answer: No, unfortunately
- Proposal: another impl. for multicore CPUs
  - Utilize (shared) cache more efficiently
  - Dynamic scheduling by runtime system
- Result: New impl. achieves better scalability on multicore CPUs

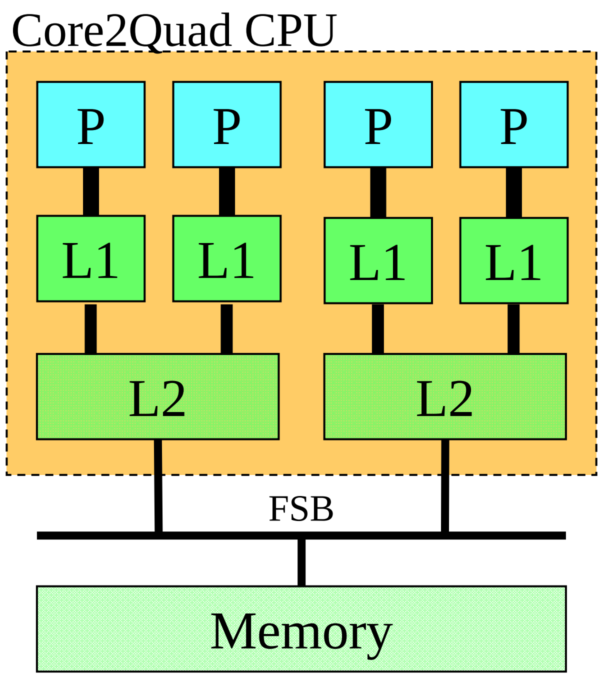
Multicore CPUs

- Trend toward multicore/manycore CPUs
  - Limitation by law of physics
  - Gain higher performance/energy ratio
  - Achieving performance by parallelism

- Multicore CPUs are now widely available
  - Intel: Core2 Duo/Quad, Xeon
  - AMD: Athlon 64 X2, Phenom, Opetoron

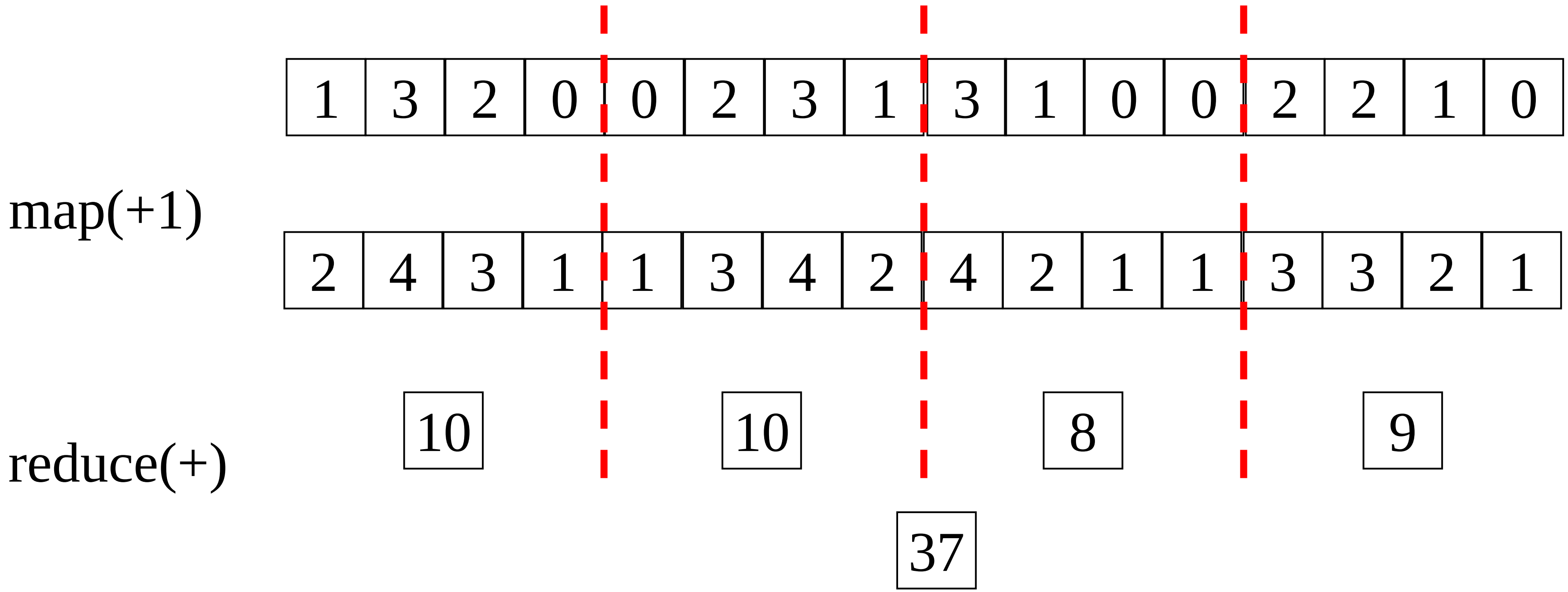
Difficulties on Multicore CPUs

- Parallelism is necessary
  - So far, not a big problem for small number of cores
  - Even for 4-core CPUs, parallelism gains performance
- More complicated cache
  - Gap between CPU speed and memory bandwidth
  - Shared-cache architecture
    - ✓ L2 in Intel Core2
    - ✓ L3 in AMD Phenom



Implementation of Skeletons for Distributed-Memory Computers

- Divide an array into segments of equal size
- Distribute them to processors
- Compute independently in parallel



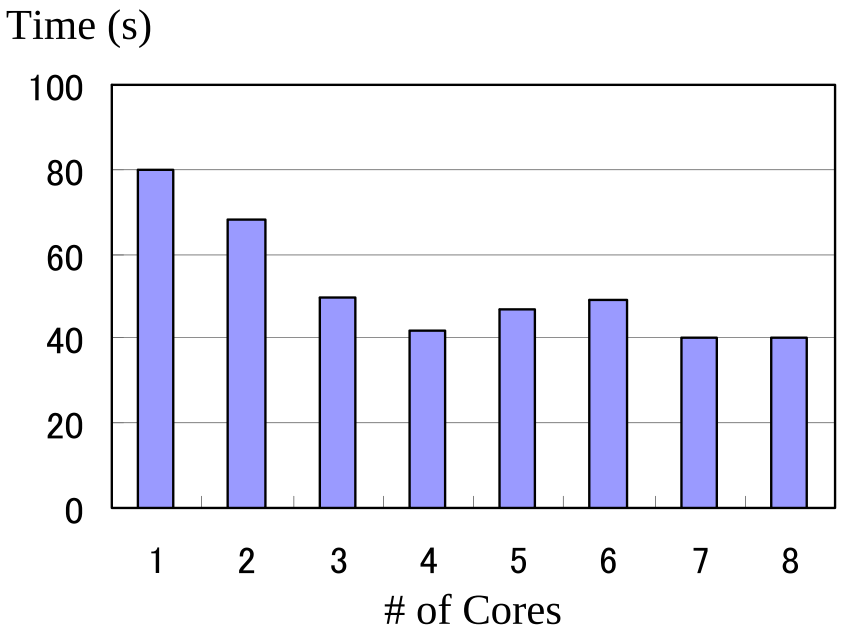
Speedup on Multicore CPUs

- Example: Apply the map skeleton 200 times

Corresponding sequential code

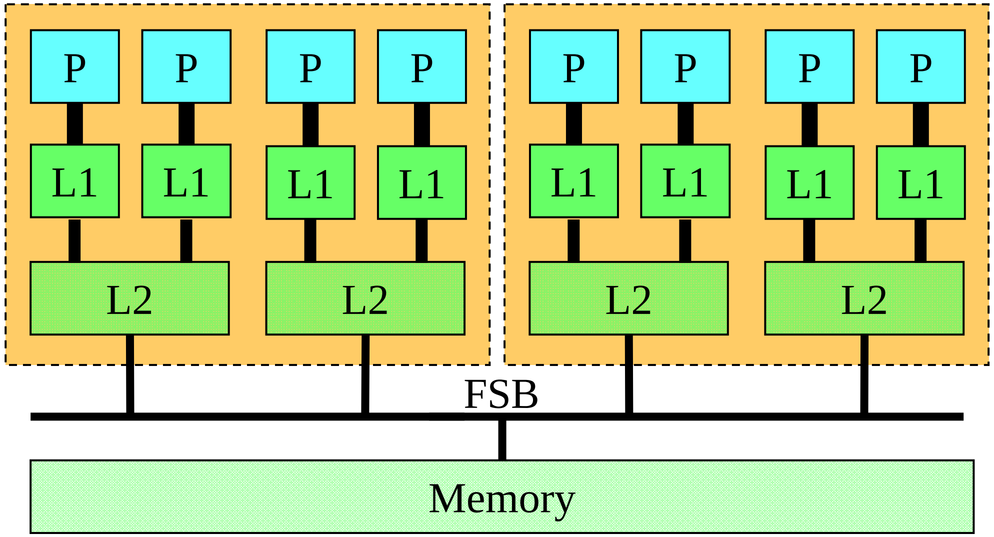
```
For (c = 0; c < 200; ++c) {
    for (i = 0; i < 200,000,000; ++i) {
        a[i] += 1; } }
```

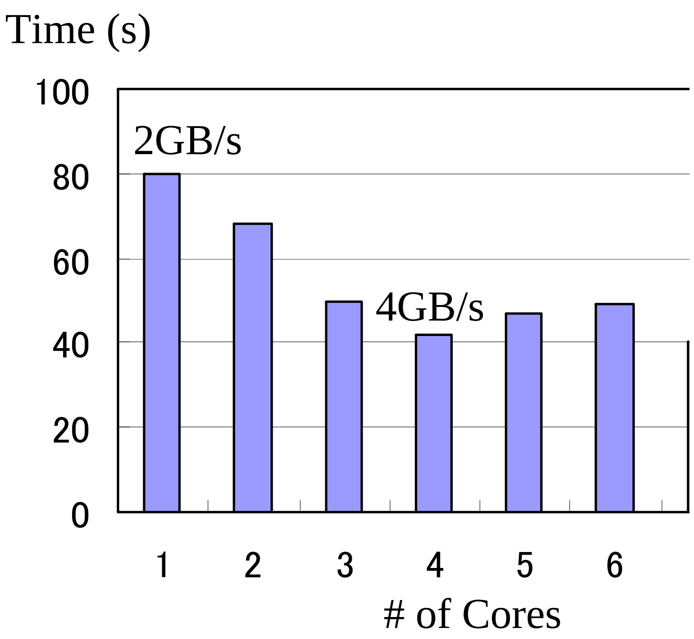
- Environment:
  - 2x quadcore Xeon CPUs
  - 8x 1GB Memory
- Not enough speedup
  - 2 times with 8 cores
  - No speedup over 4 cores



What is the Problem?

- Bandwidth of memory is insufficient (saturated)
  - Requires more than 4GB/s in total

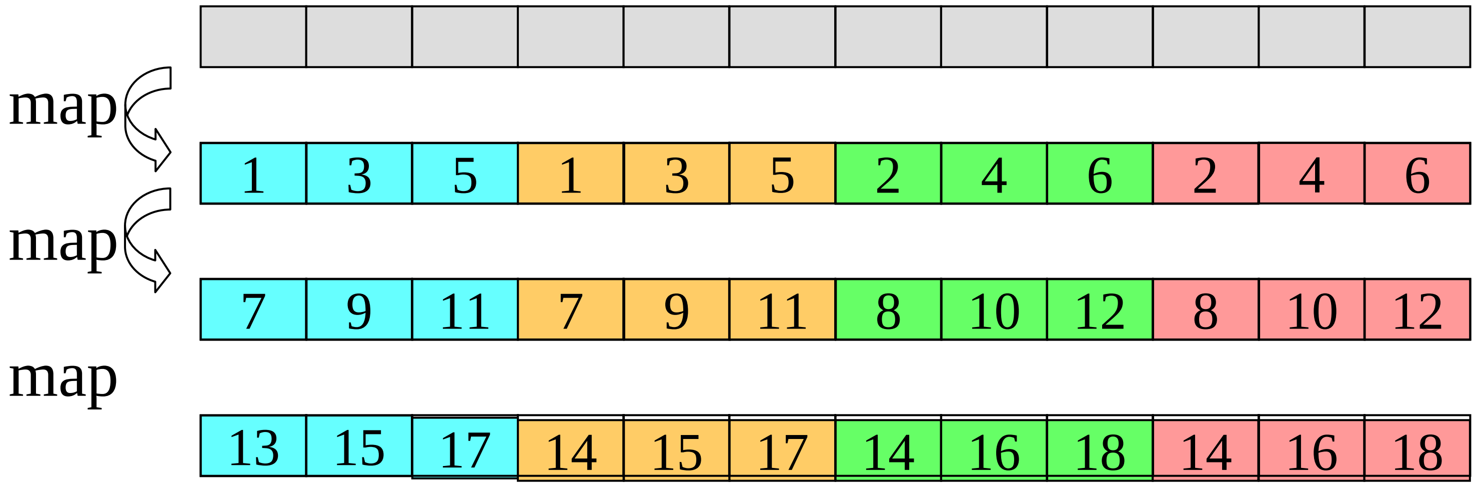




➔ Utilize cache and avoid memory access

Illustrating Execution (1)

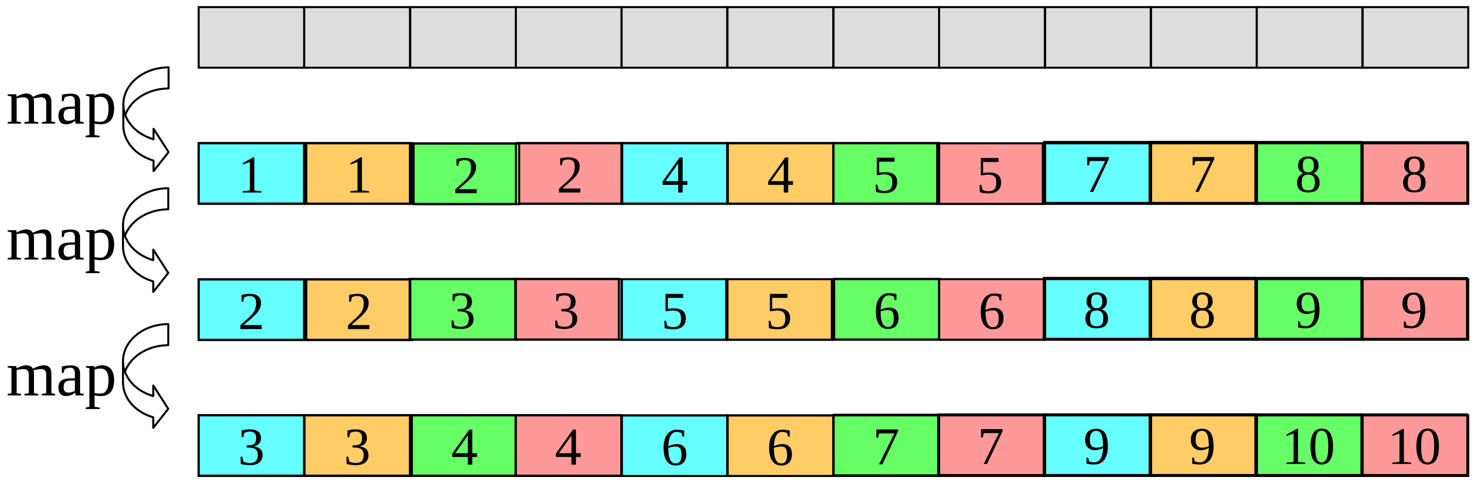
- Naive Implementation
  - Memory bandwidth = 2
  - Size of block = cache-size / 4



  - The speedup achieved is only 2 (=36/18)

Illustrating Execution (2)

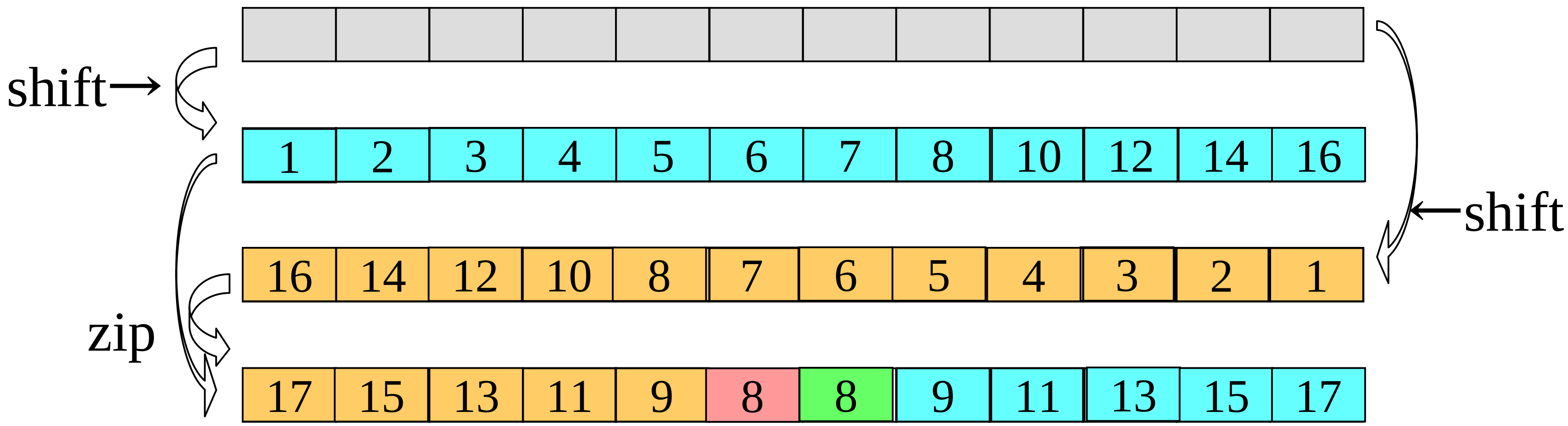
- Cache-efficient implementation
  - Memory bandwidth = 2
  - Size of block = cache-size / 4



  - Speedup achieved is 3.6 (=36/10)

Illustrating Execution (3)

- Execution with a runtime scheduling
  - Consider the case that dependencies exist
  - We can achieve some speedups even such a case



Overview of the Framework

- Program: Write it using skeletons
- Skeletons: C++ templates and function objects
  - Generate a dependency graph of skeletons
  - Split data into small blocks smaller than cache size
- Runtime: Scheduling computation on blocks
  - Implementation: On pthread library
  - Select a block whose input is given
  - Scheduling Policy: Queue (FIFO) / Priority queue

Program Code

- Almost the same as the usual skeleton programs
  - Inside of loop: update by $x^{t+1}[i] = c * x^{t}[i] + d * x^{t}[i-1]$

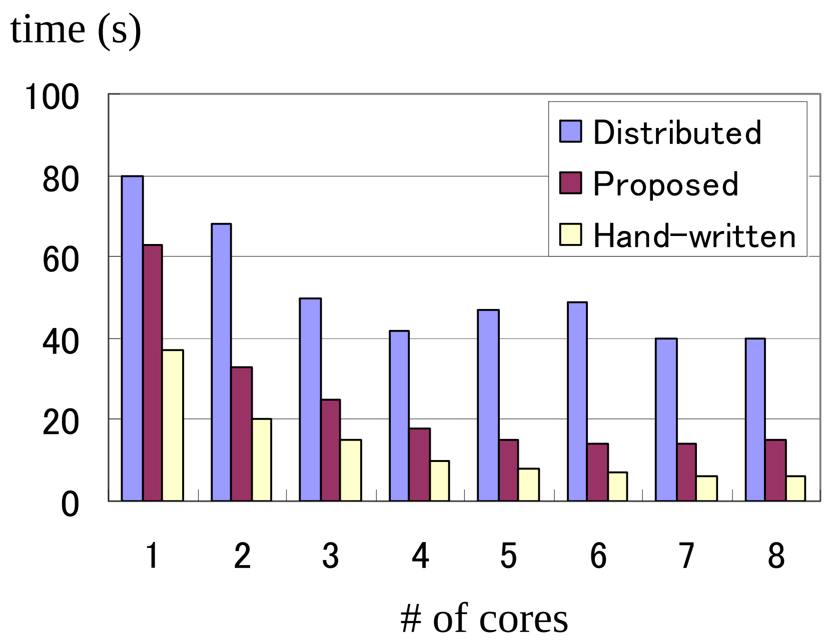
```
skel<double>* as = mcs::generator(zero, N);
skel<double>* as_left;
for (int t = 0; t < count; t++) {
    as_left = mcs::shift(c, as);
    as = mcs::zipwith(f, as, as_left);
}
as->eval();
```

  - The skeletons just generate dependency graph and eval() perform the actual computation.

Experiments: Map 200 times

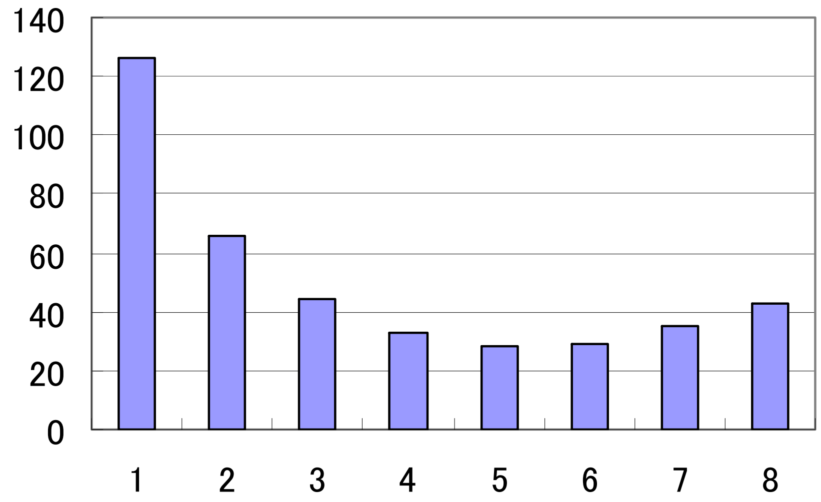
- Compared the system with two others
  - Distributed: implementation for distributed-memory
  - Hand-written: A specialized one using L1 cache

- Results
  - Speedup up to 6 cores
  - 3 times faster
    on 6 cores or over
  - Some overhead
    - ✓ lock in pthread



Experiments: Differential Equation

- Simulation of simple differential equation
  - 100M elements
  - 200 updates by $x^{t+1}[i] = c * x^t[i] + d * x^t[i-1]$
  - The code presented before
- Result
  - Speedup up to 5 cores (almost linear)
  - Some overhead again



Conclusion

- Classic implementation of data-parallel skeletons may be not efficient on multicore CPUs
  - Due to shared resource (memory bandwidth)
  - Utilizing cache hierarchy is necessary

- Proposed an efficient implementation of data-parallel skeletons on multicore CPUs
  - By runtime scheduling system
  - Prototype implemented with C++ templates
  - Good scalability for several applications

Ongoing and Future Work

- Developing framework/runtime-system for cluster of multicore PCs
  - How to divide into independent tasks?
  - Mixture of message-passing and threads
- Dynamic scheduling vs. Static scheduling
  - Current runtime system has (not small) overhead
  - Can we develop good scheduling from a DAG of restricted shape (by skeletons)?