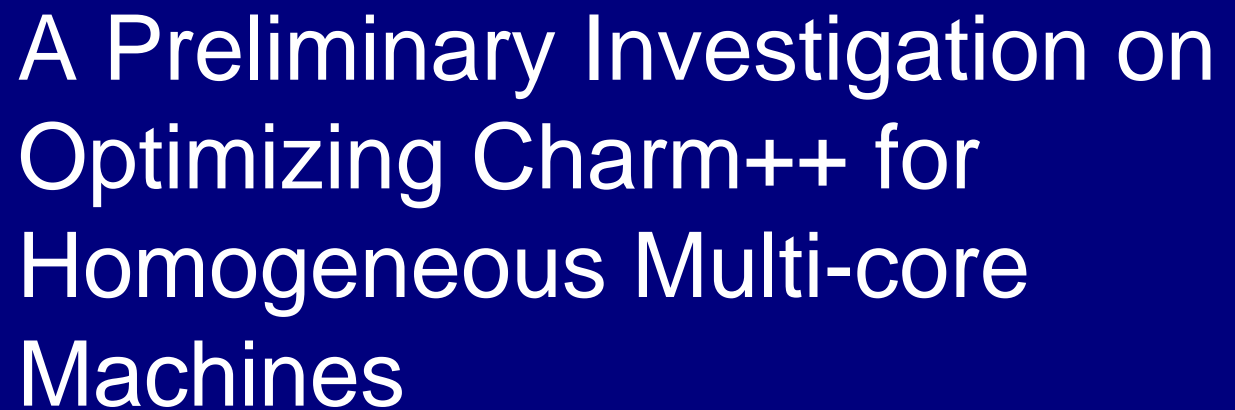


Chao Mei

05/02/2008

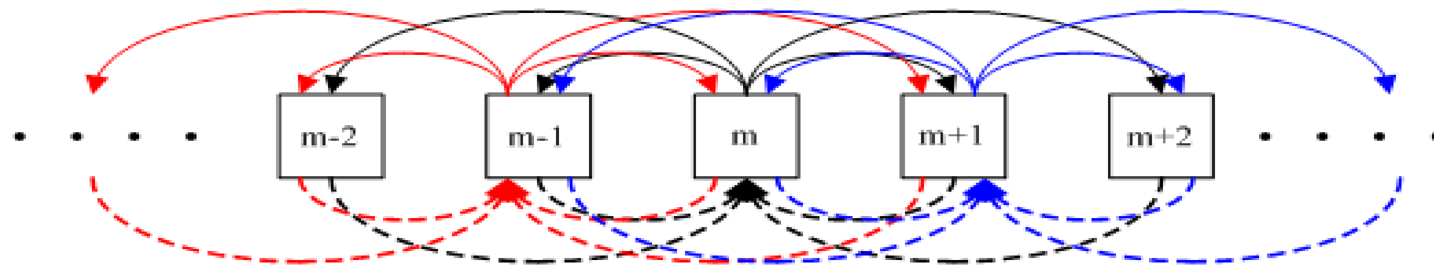
The 6th Charm++ Workshop

Motivation

- Clusters are built from multicore chips
  - 4 cores/node on BG/P
  - 8 cores/node on Abe (2 Intel quad-core chips)
  - 16 cores/node on Ranger (4 AMD quad-core chips)
  - …
- Charm has a building version for SMP node for many years
  - Not tuned
- So, what are the issues for getting high performance?

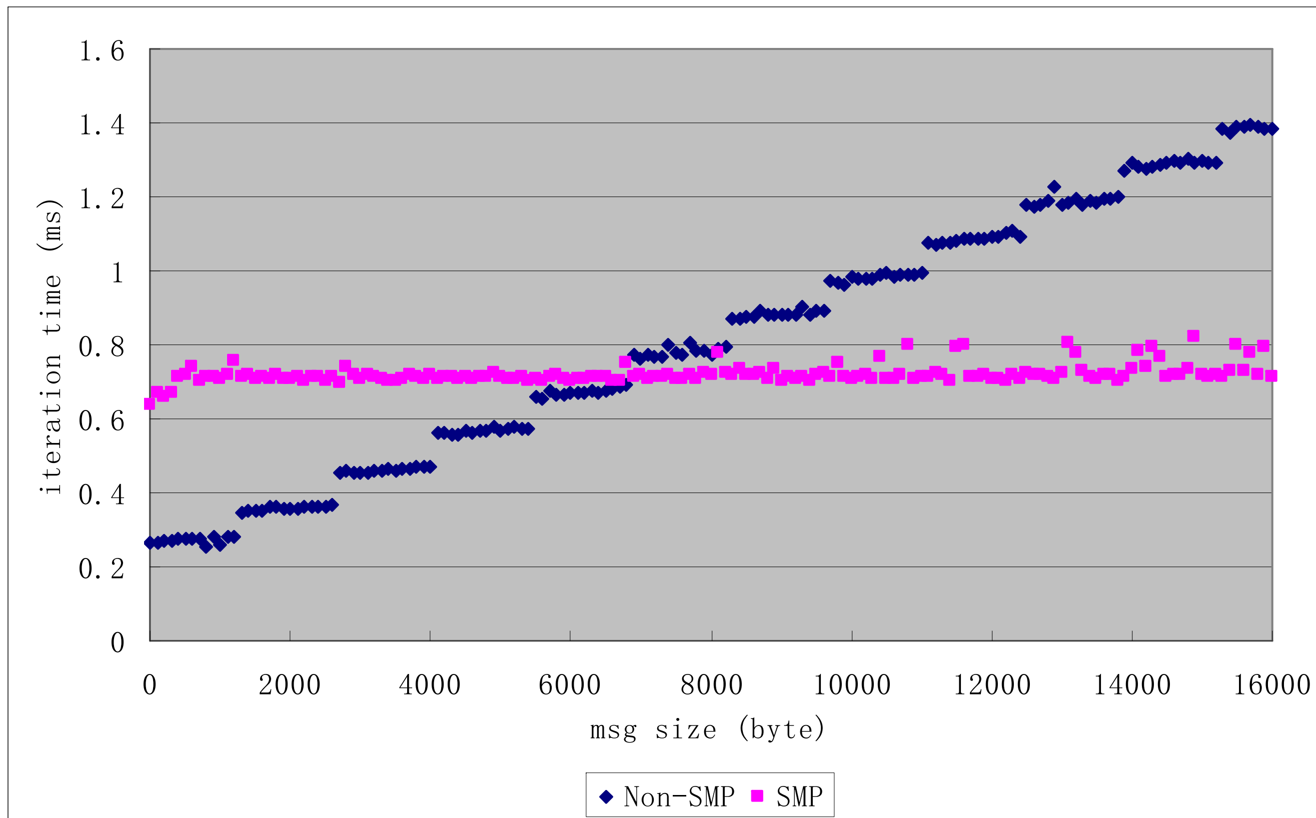
Start with a kNeighbor benchmark

- A synthetic kNeighbor benchmark
  - Each element communicates with its neighbors in K-stride (wrap-around), and then neighbors send back an acknowledge.
  - An iteration: all elements finish the above communication



- Environment
  - A smp node with 2 Xeon quadcores, only use 7 cores
  - Ubuntu 7.04; gcc 4.2
  - Charm: net-linux-amd64-smp vs. net-linux-amd64
  - 1 element/core, K=3

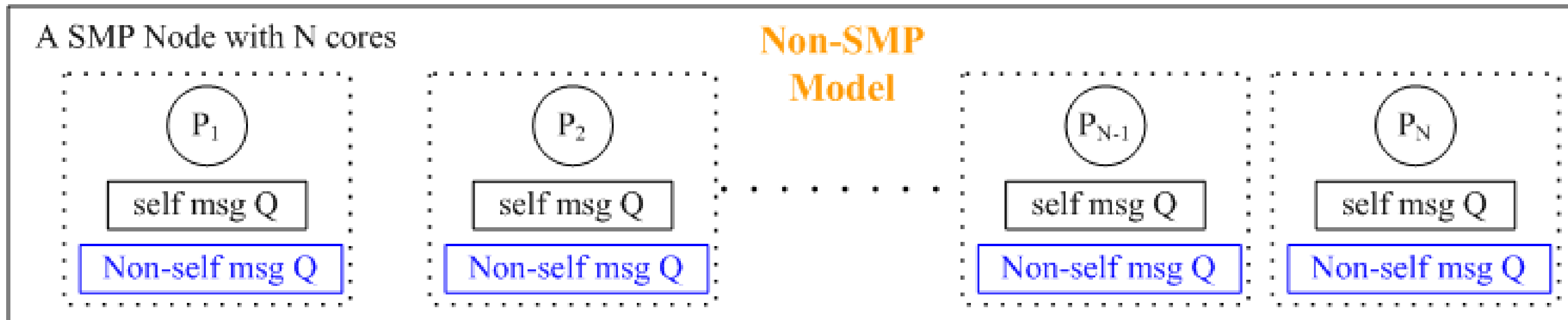
Performance at first glance

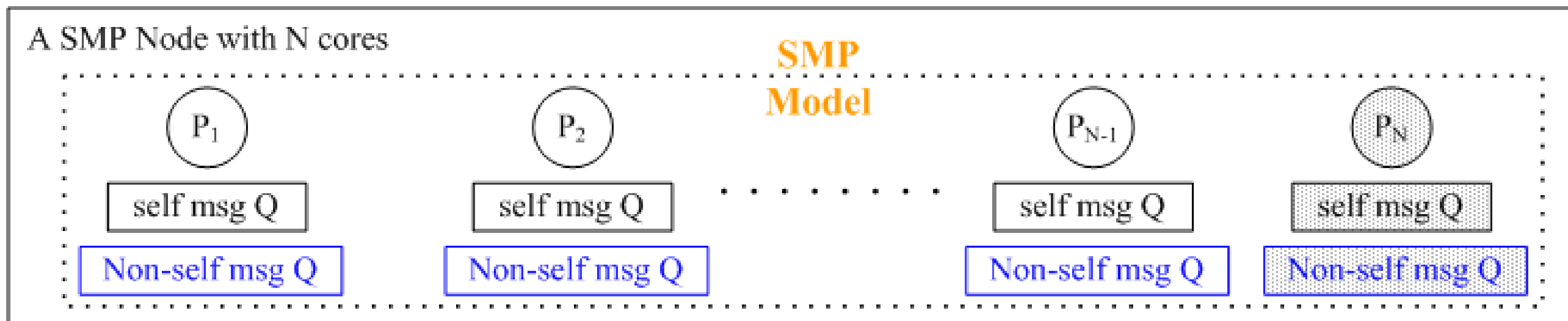


Outline

- Examine the communication model in Charm++ between the Non-SMP and SMP layers
- Describe current optimizations for SMP step by step
- Talk about a different approach to utilize multicore
- Conclude with the future work

Communication model for the multicore





For inter-core communication inside a node:
1. SMP: message is passed via memory pointer
2. Non-SMP: message is passed through NIC

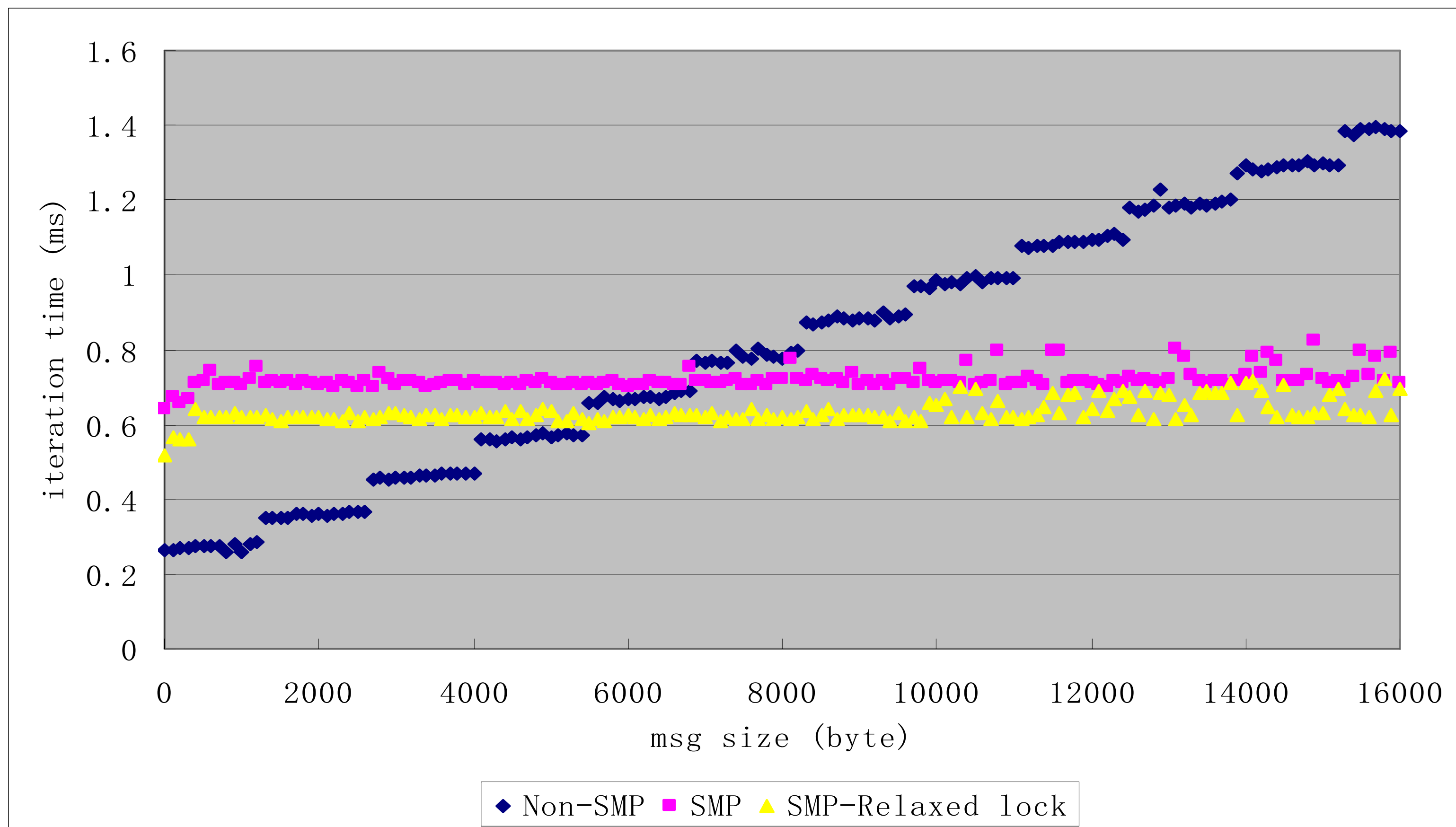
Chao Mei (chaomei2@uiuc.edu)
Parallel Programming Lab, UIUC

Possible overheads in SMP version

- **Locks**
  - Overusing locks to ensure correctness
  - Locks in message queues
  - …
- **False sharing**
  - Some per thread data structures are allocated together in an array form: e.g. each element in "CmiState state[numThds]" belongs to a thread

Reducing the usage of locks

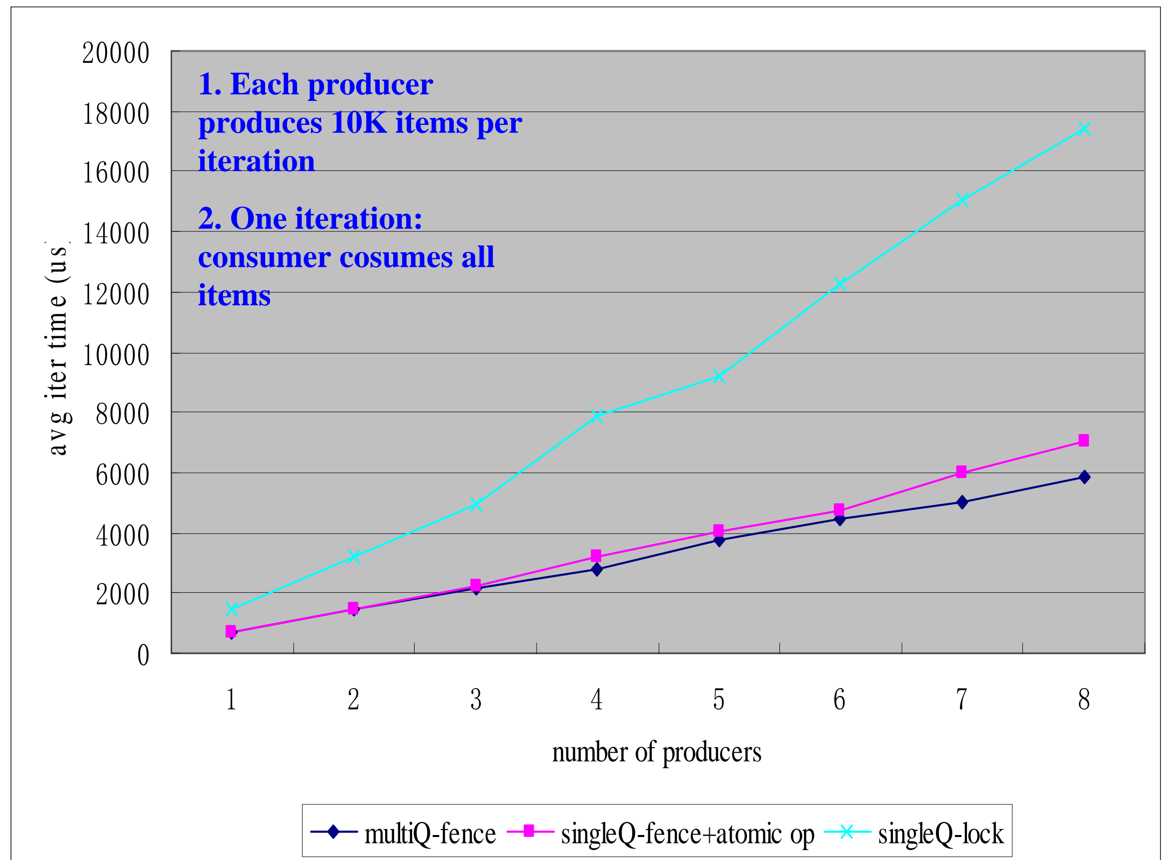
- **By examining the source codes, finding overuse of locks**
  - Narrower sections enclosed by locks



Chao Mei (chaomei2@uiuc.edu)
Parallel Programming Lab, UIUC

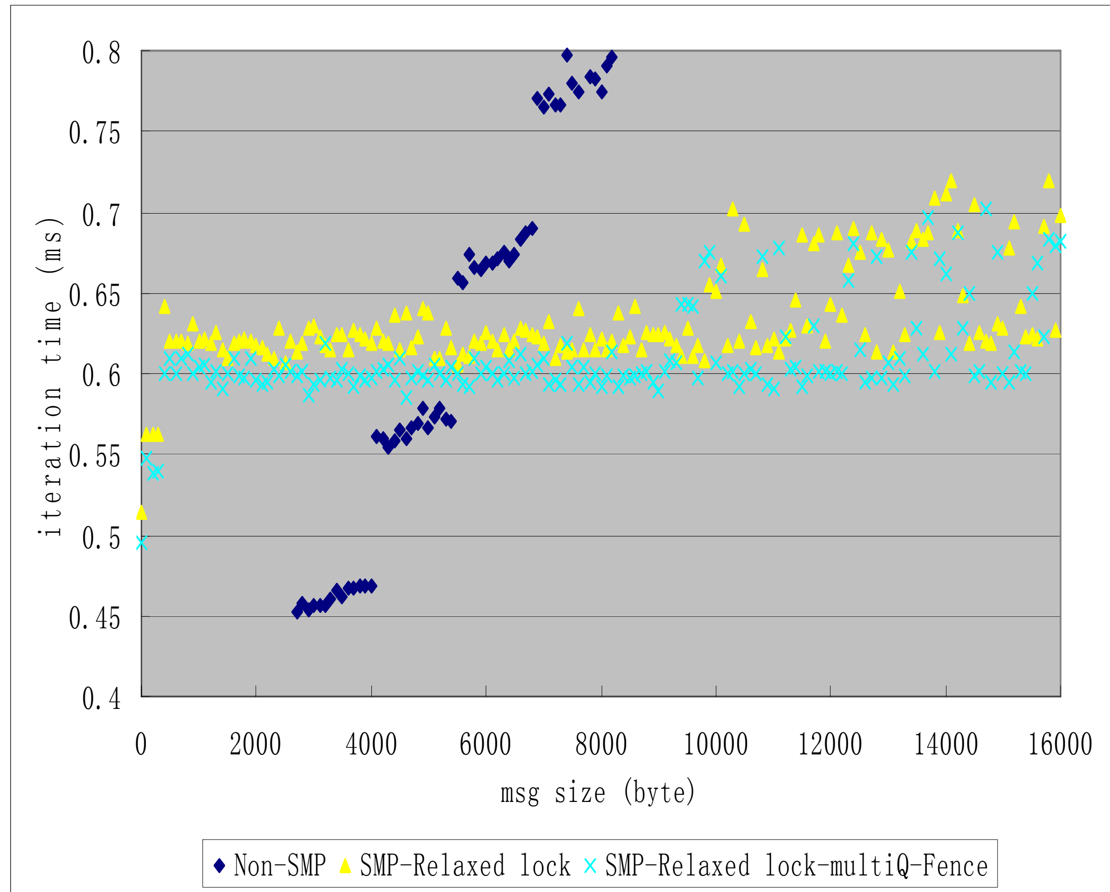
Overhead in message queues

- A micro-benchmark to show the overhead in message queues
  - N producers, 1 consumer
  - lock vs. memory fence + atomic operation (fetch-and-increment)
  - 1 queue vs. N queues



Applying multi Q + Fence

- **Less than 2% improvement**
  - **Much less contention compared with the micro-benchmark**



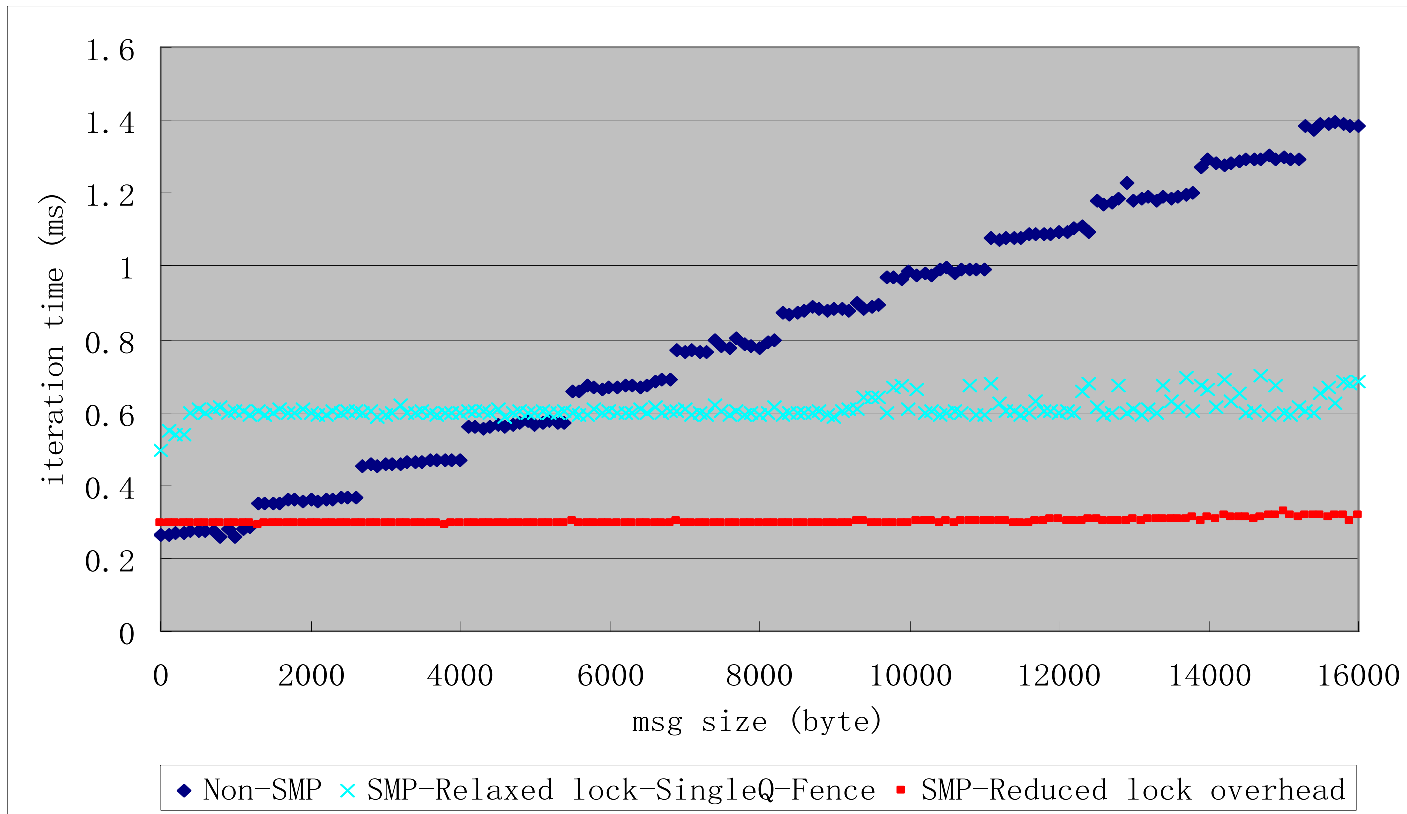
Big overhead in msg allocation

- **We noticed that:**

msg creation time				
	1k-byte msg		10k-byte msg	
iter	smp	nonsmp	smp	nonsmp
1	6	2	4	1
2	5	1	5	1
3	5	1	4	1
4	3	1	3	2
5	4	1	5	1
6	7	1	3	3

- **We used our own default memory module**
  - Every memory allocation is protected by a lock
  - Provide some useful functionalities in Charm++ system (a historic reason not using other memory modules)
    - memory footprint information, memory debugger
    - Isomalloc

Switching to OS memory module



We don't lose the aforementioned functionalities by recent updates ☺

Identifying false sharing overhead

- **Another micro-benchmark**
  - Each element repeatedly sends itself a message, but each time the message is reused (i.e., not allocating a new message)
  - Benchmark timing of 1000 iterations

- **Use Intel VTune performance analysis tool**
  - Focusing on the cache misses caused by "Invalidate" in the MESI coherence protocol

- **Declaring variables with "__thread" specifier will make them thread private**

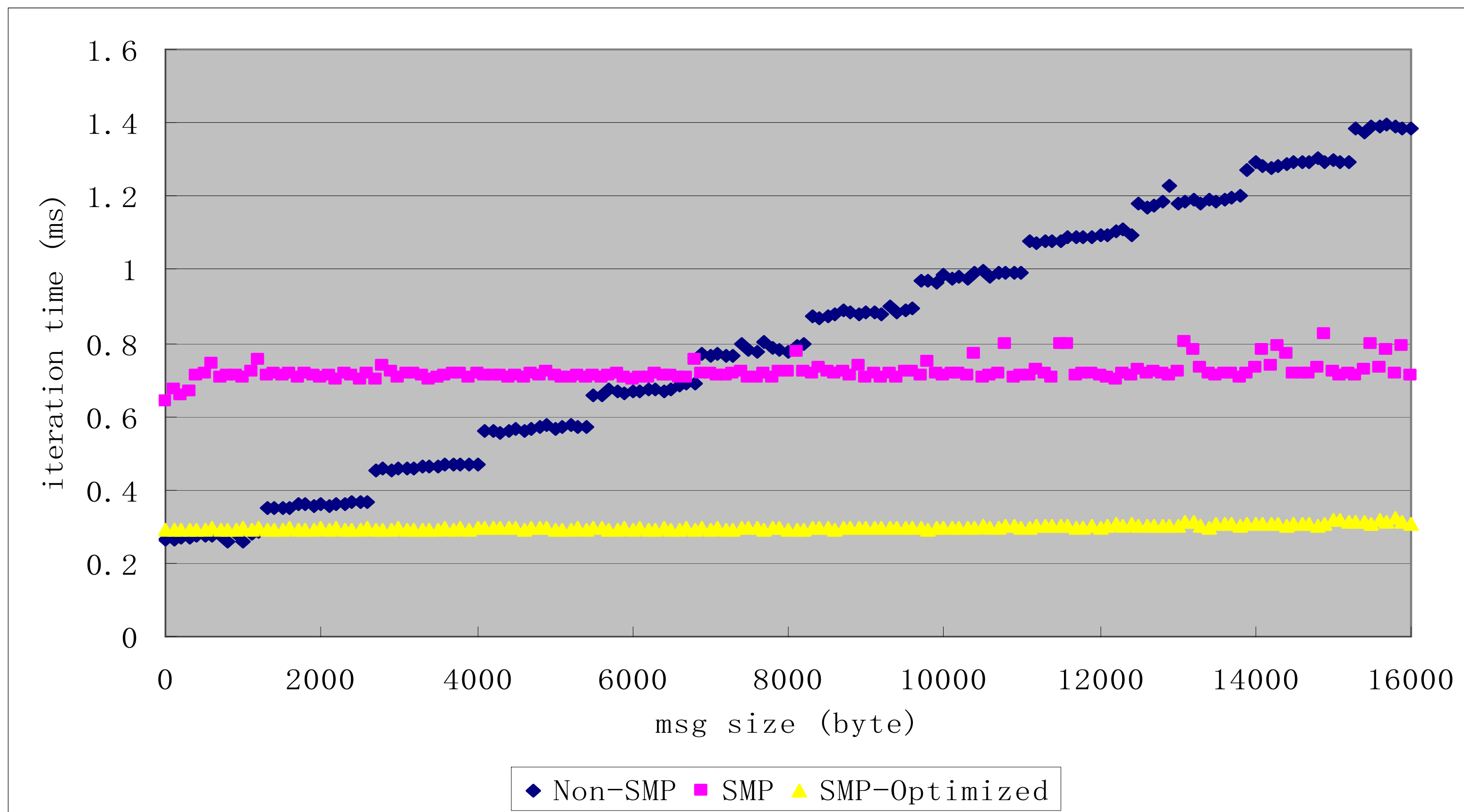
Performance for the micro-benchmark

- **Parameters: 1 element/core, 7 cores**
- **Before: 1.236 us per iteration**
- **After: 0.913 us per iteration**

Chao Mei (chaomei2@uiuc.edu)
Parallel Programming Lab, UIUC

Adding the gains from removing false sharing

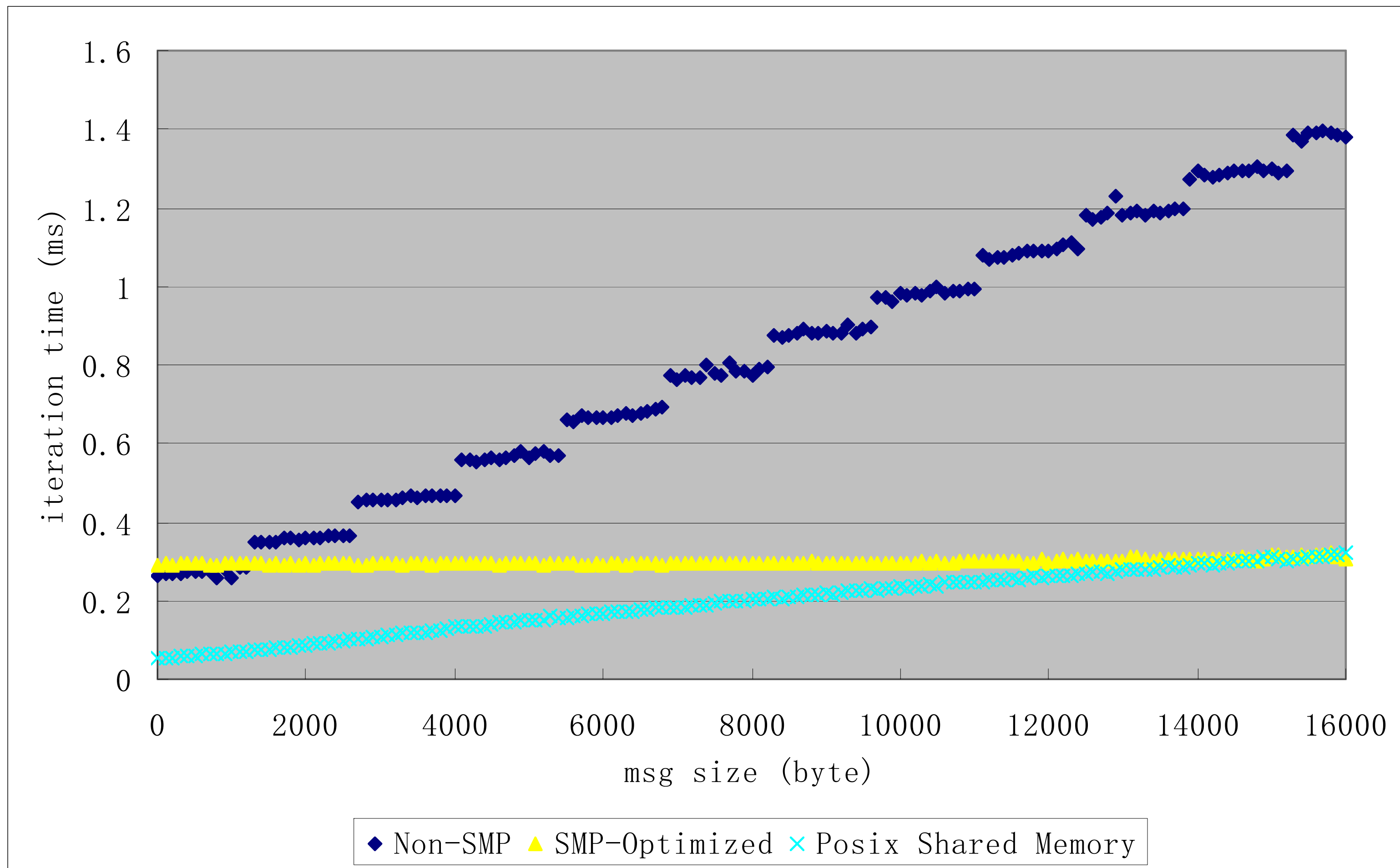
- ***Around 1% improvement***



Rethinking communication model

- **Posix-shared memory layer**
- **No threads, every core still runs a process**
- **Inter-core message passing doesn't go through NIC, but through memory copy (inter-process communication)**

Performance comparison



Future work

- **Other platform**
  - **BG/P**
- **Optimize the posix shared memory version**
- **Effects on real applications**
  - **For NAMD, initial result shows that SMP helps up to 24 nodes on Abe**
- **Any other communication models**
  - **Adaptive one?**