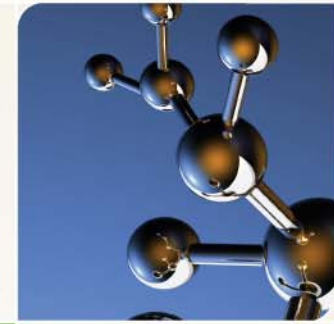
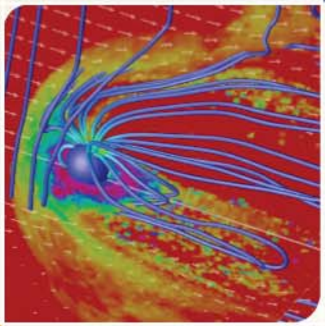
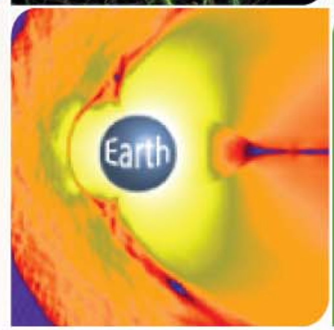
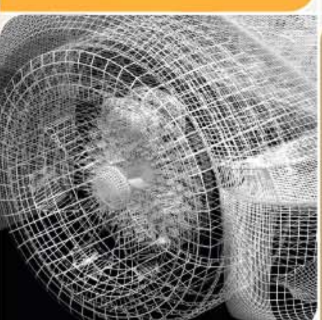






SWARM: A Parallel Programming Framework for Multicore Processors



David A. Bader, Varun N. Kanade and Kamesh Madduri

Georgia Tech College of Computing
Computational Science and Engineering

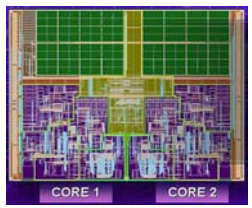
Our Contributions

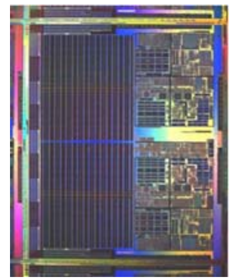
- SWARM: SoftWare and Algorithms for Running on Multicore, a portable open-source parallel framework for multicore systems
  - http://multicore-swarm.sourceforge.net/
- A library of fundamental parallel primitives: prefix-sums, list ranking, sorting and selection, symmetry breaking etc.
- Computational model for analyzing algorithms on multicore systems

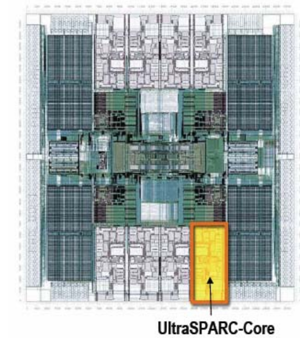
Talk Outline

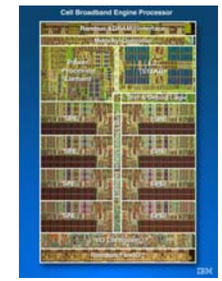
- Introduction
- Model for algorithm design on multicore systems
  - Case studies
  - Performance
- SWARM
  - Programming framework
  - Algorithm design
  - Performance

Multicore Computing









High-performance multicore programming requirements

- Exploit concurrency at the algorithmic level, and design efficient parallel algorithms
- Judiciously utilize memory bandwidth
- minimize inter-processor communication, synchronization

Multicore Computing

Model for multicore algorithm design

SWARM: A Parallel Programming Framework

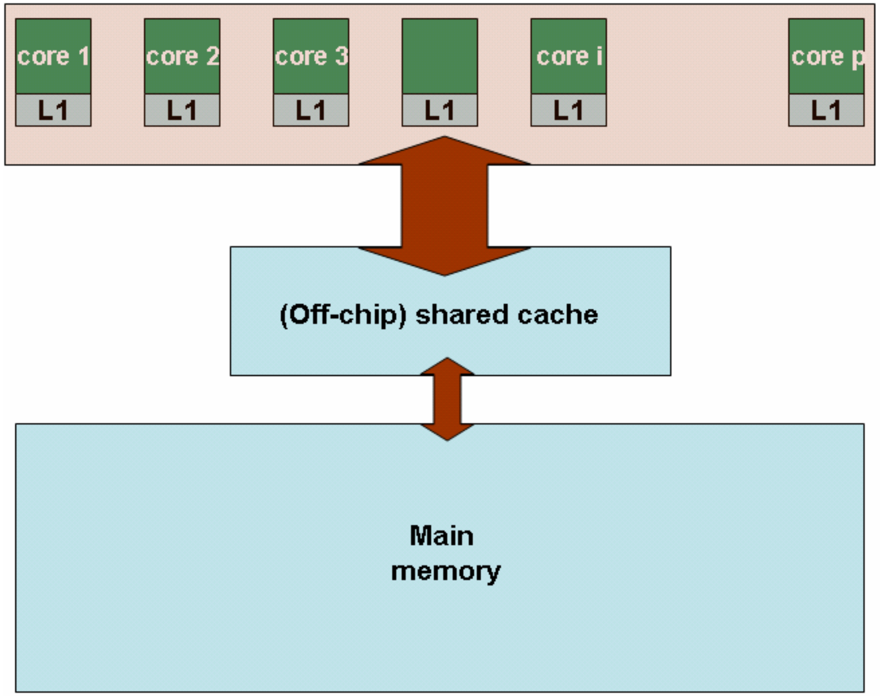
High-performance multicore programming requirements

- Exploit concurrency at the algorithmic level, and design efficient parallel algorithms
- Judiciously utilize memory bandwidth
- minimize inter-processor communication, synchronization

Multicore Algorithm Design

- Architectural model: *p* homogeneous processing cores, dedicated L_1 cache, shared L_2 cache
- Memory Bandwidth of

  L_1 cache > L_2 cache > main memory



Complexity Model

Algorithm complexity is given by

- Computational Complexity: $T_c(n,p) = \max_i T_{ci}(n,p), i = 1,\ldots,p$
  - RAM model of computation, complexity as a function of input size
- Memory accesses: $T_M(n) = B(n)\sigma^{-1}$
  - B: no. of blocks transferred from main memory to shared L2 cache, σ: bandwidth parameter
  - Aggarwal-Vitter I/O model of computation
- Synchronization: $T_s(n) = S(n)L$
  - S(n): complexity of synchronization operations (barriers, locks), L: synchronization overhead parameter

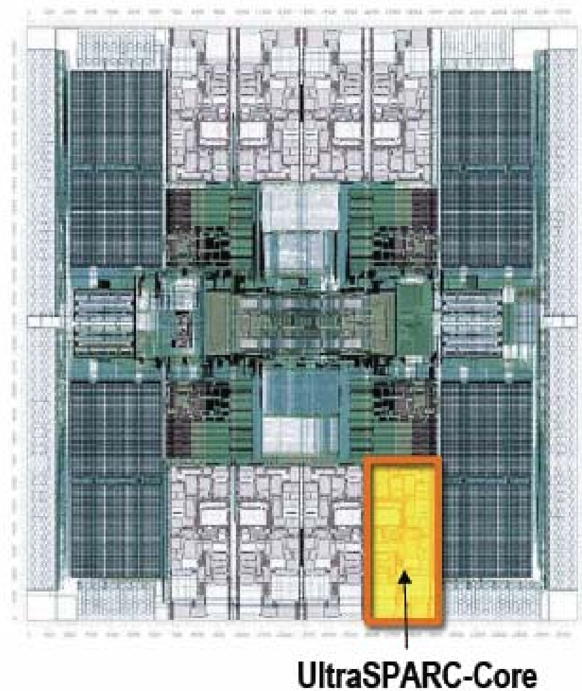
Multicore Algorithm Design

- Complexity is given by the tuple $<T_{C,} T_{M,} T_S>$
- Cache-aware approaches
  - Data layout optimizations, blocking/tiling, padding
  - Merge Sort case-study
- Cache-oblivious algorithms
- Minimize synchronization overhead
  - Lock-free algorithms, atomic operations

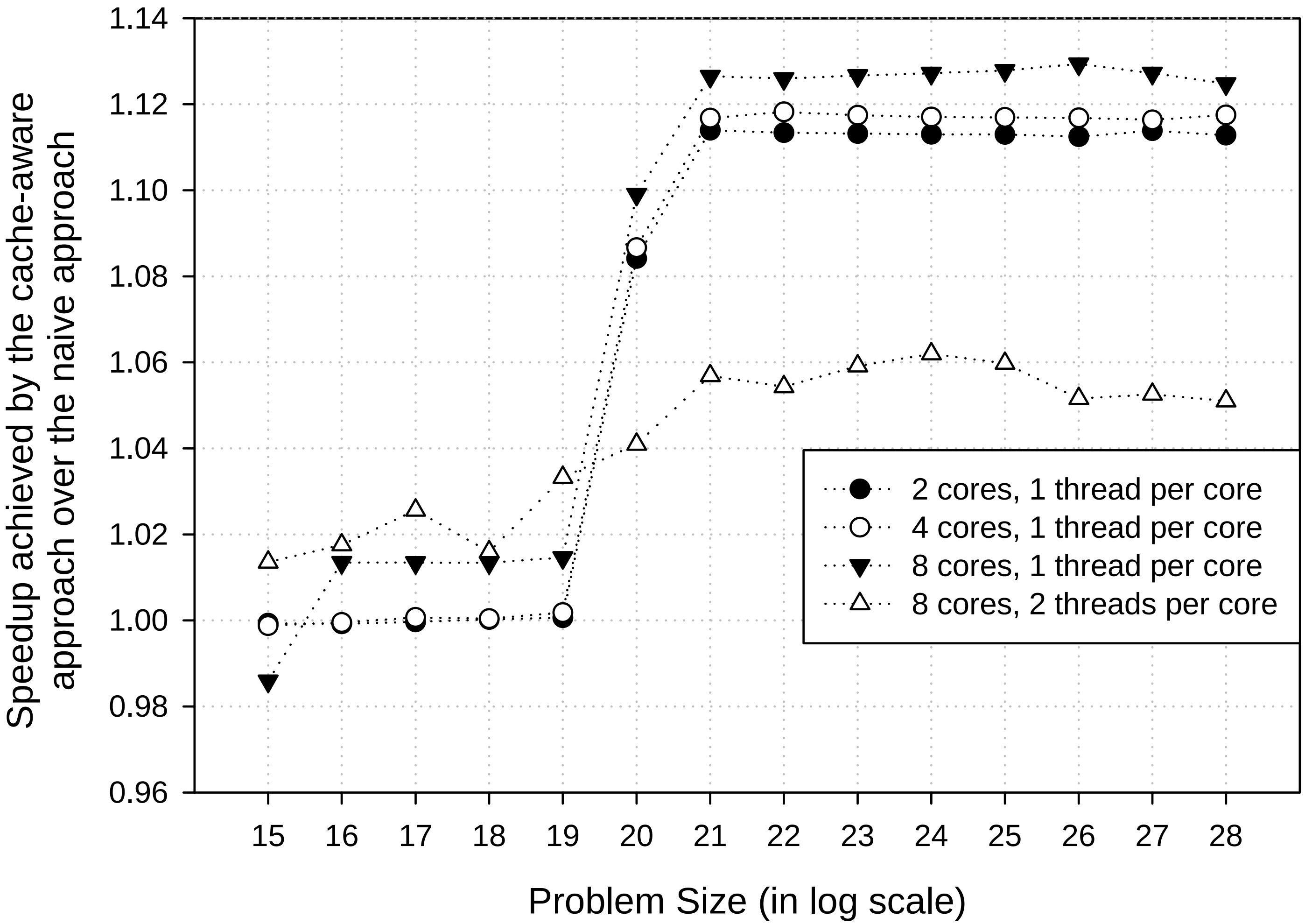
- All these paradigms considered in SWARM parallel primitives and library design

Test Platform

- Sun Fire T2000 (UltraSparc T1 Niagara processor)
  - 8 multithreaded cores
  - 4 threads per core
  - Shared 3MB on-chip L2 cache
  - 1.0GHz clock speed



Tiling optimization benchmark: Performance on Sun Fire T2000



Georgia Tech College of Computing

SWARM

- Multicore programming framework and a collection of multicore-optimized libraries
- Portable, open-source
  - http://multicore-swarm.sourceforge.net/
  - Current: version 1.1
- Examples and efficient implementations of various parallel programming primitives
  - Prefix-sums, pointer-jumping, divide and conquer, pipelining, graph algorithms, symmetry breaking

SWARM

- POSIX-threads based framework
- Support for parallel primitives
  - data parallel, control, memory management
  - barrier, replicate, scan, broadcast
- Incremental approach to parallelizing applications
  - Minimal modifications to existing code

Typical SWARM Usage

- C code

```
int main (int argc, char **argv) {
        SWARM_Init(&argc, &argv);
        /* sequential code */
        ....
        ....

        fun1(a, b);
        SWARM_Run((void *) fun1(a, b));

        /* more sequential code */
        fun2(c,d);
        ::::

        SWARM_Finalize();
}
```

Identify compute-intensive functions

Parallelize with SWARM library

Data parallelism: pardo directive

- pardo: *Parallel do*, implicitly partitions a loop among the cores without the need for coordinating.
- SWARM provides both block and cyclic partitioning options

```
/* pardo example: partitioning a
"for" loop among the cores */
pardo(i, start, end, incr) {
    A[i] = B[i] + C[i];
}
```

Control

- SWARM control primitives restrict threads that can participate in a context.

```
// THREADS: total number of execution threads
// MYTHREAD: the rank of a thread, from 0 to
// THREADS-1

/* example: execute code on a specific thread */
on_thread (3) {
      ....
      ....
}

/* example: execute code on just one thread */
on_one_thread {
      ...
      ...
}
```

Memory management

- SWARM provides two directives
  - SWARM_ malloc: dynamically allocate a shared structure
  - SWARM_free: release shared memory back to the heap

```
/* example: allocate a shared array of size n */
A = (int *)SWARM_malloc(n*sizeof(int),TH);

/* example: free the array A */
SWARM_free(A);
```

Synchronization

- Barrier: `SWARM_Barrier()`
  - Two variants
- Locks
  - POSIX threads Mutex locks, user-level atomic locks

Georgia Tech College of Computing

Other communication primitives

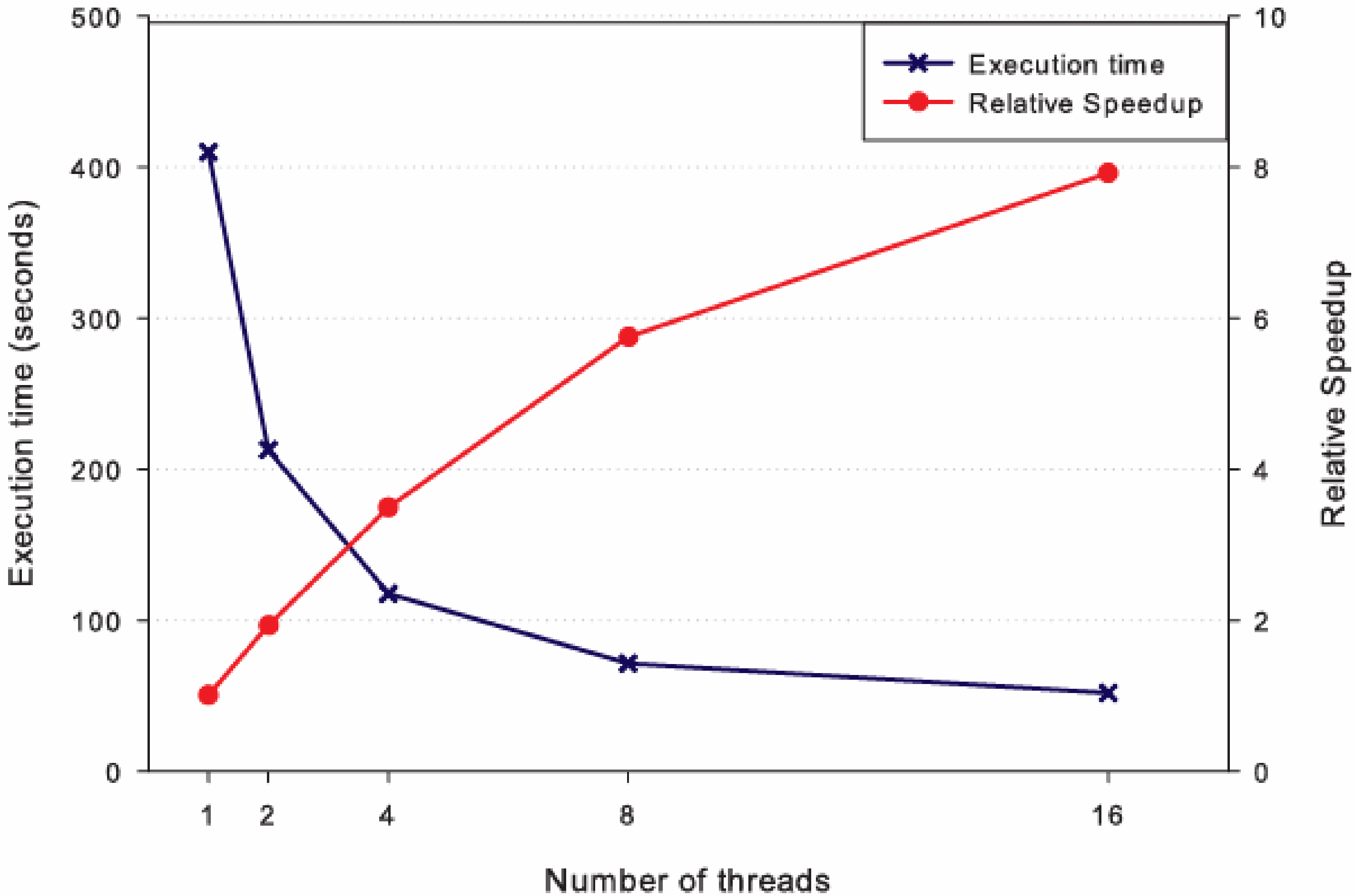
- Broadcast: supplies each processing core with the address of the shared buffer by replicating the memory address.

```
/* function signatures */
int SWARM_Bcast_i (int myval, THREADED);
int* SWARM_Bcast_ip (int* myval, THREADED);
char SWARM_Bcast_c (char myval, THREADED);
```

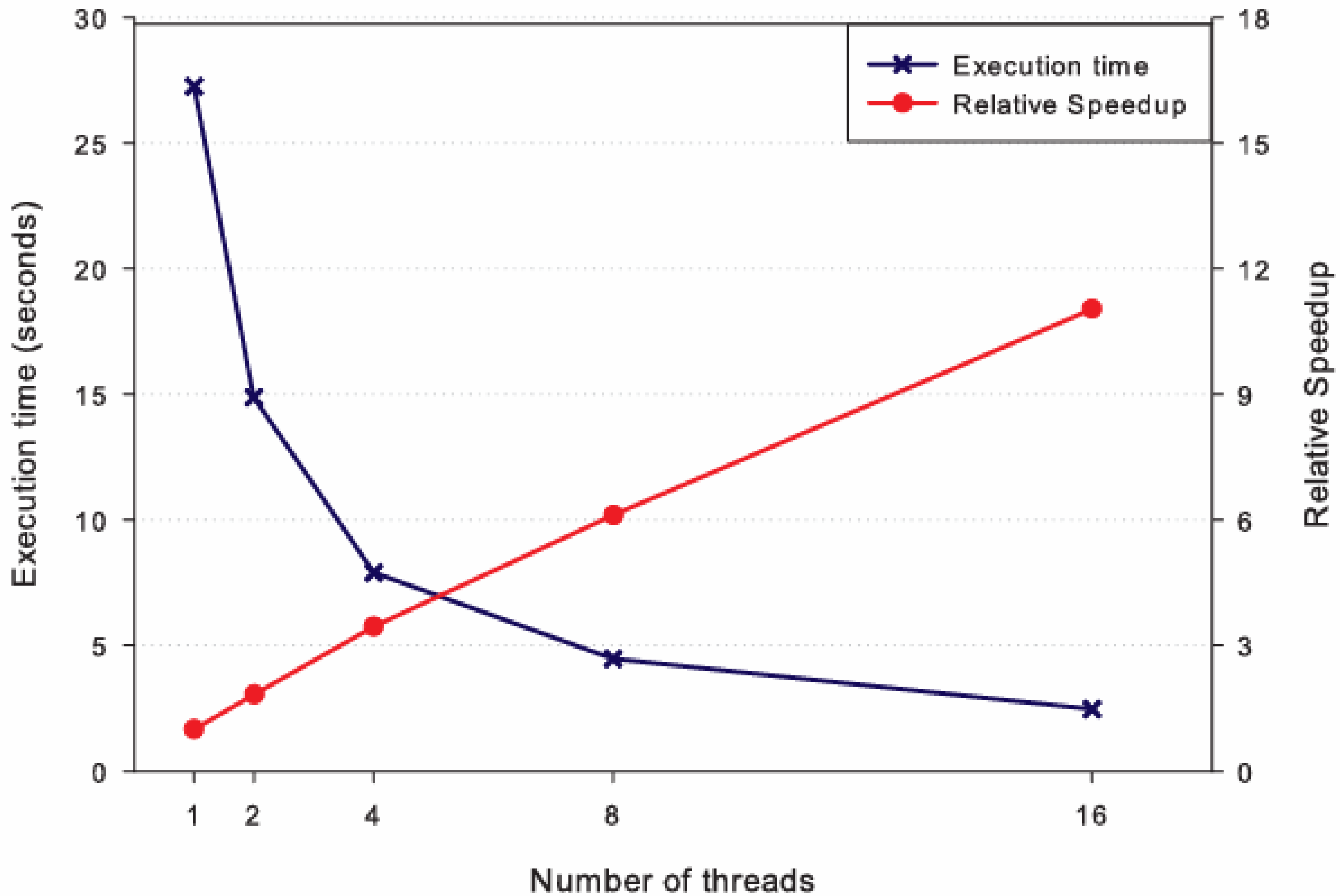
- Reduce: performs a reduction operation with a binary associative operator, such as addition, multiplication, maximum, minimum, bitwise-AND, and bitwise-OR

```
/* function signatures */
int SWARM_Reduce_i(int myval, reduce_t op,
THREADED);
double SWARM_Reduce_d(double myval, reduce_t op,
THREADED);
```

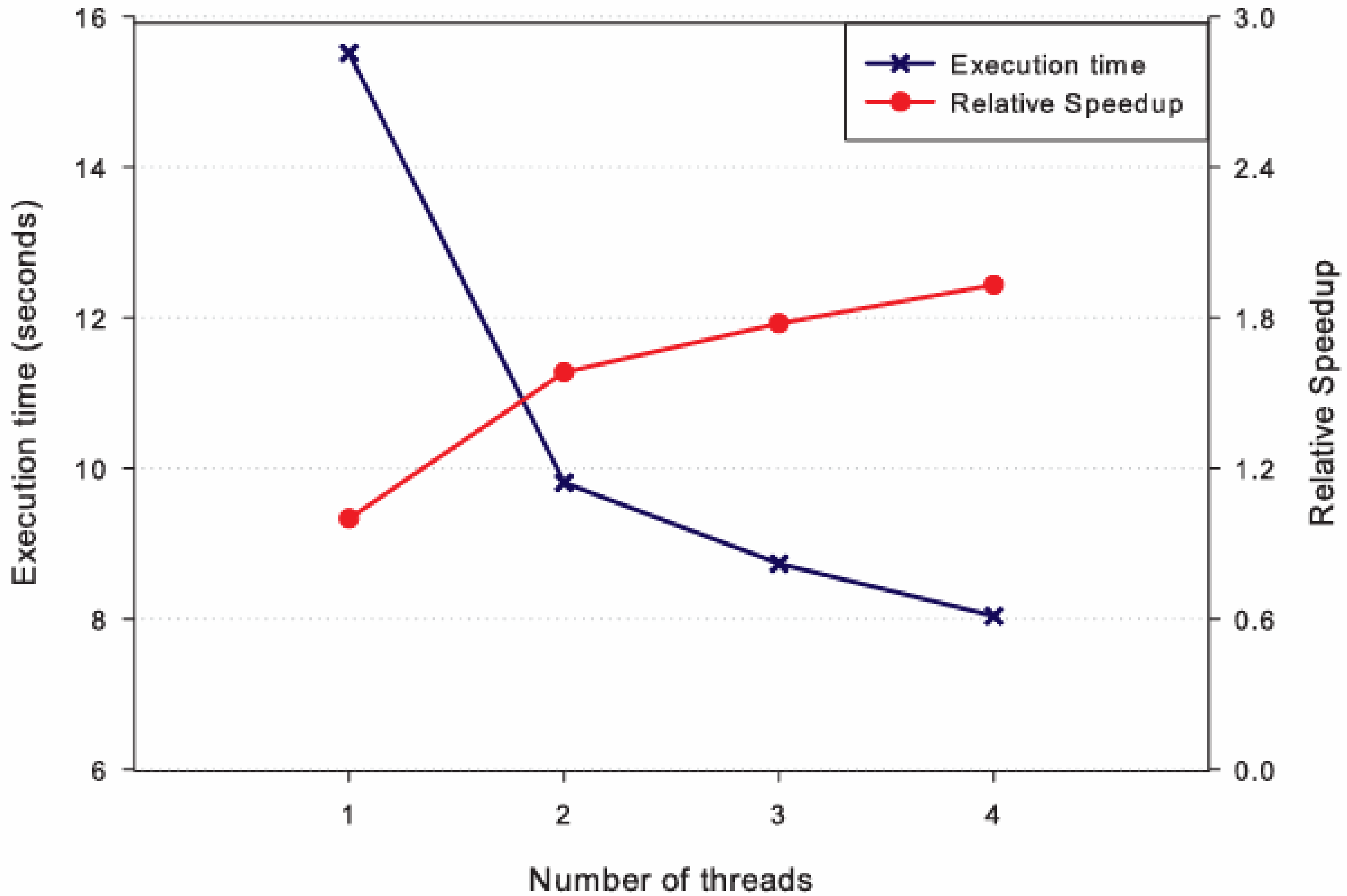
SWARM: Merge Sort Performance, Sun Fire T2000



SWARM: List Ranking Performance, Sun Fire T2000



SWARM: List Ranking Performance, Intel dual-core Xeon 5150



Acknowledgment of Support

- National Science Foundation
  - **CSR:** A Framework for Optimizing Scientific Applications (06-14915)
  - **CAREER:** High-Performance Algorithms for Scientific Applications (06-11589; 00-93039)
  - **ITR:** Building the Tree of Life -- A National Resource for Phyloinformatics and Computational Phylogenetics (EF/BIO 03-31654)
  - **ITR/AP:** Reconstructing Complex Evolutionary Histories (01-21377)
  - **DEB** Comparative Chloroplast Genomics: Integrating Computational Methods, Molecular Evolution, and Phylogeny (01-20709)
  - **ITR/AP(DEB):** Computing Optimal Phylogenetic Trees under Genome Rearrangement Metrics (01-13095)
  - **DBI:** Acquisition of a High Performance Shared-Memory Computer for Computational Science and Engineering (04-20513).
- IBM PERCS / DARPA High Productivity Computing Systems (HPCS)
  - DARPA Contract NBCH30390004
- IBM Shared University Research (SUR) Grant
- Sony-Toshiba-IBM (STI)
- Microsoft Research
- Sun Academic Excellence Grant

Conclusions

- SWARM: SoftWare and Algorithms for Running on Multicore, a portable open-source parallel framework for multicore systems
  - http://multicore-swarm.sourceforge.net/
- We present a complexity model for algorithm design on multicore systems
  - It is critical to optimize memory access patterns and synchronization on multicore systems
- Future work: more SWARM libraries and primitives