Pthreads for Dynamic and Irregular Parallelism

Girija Narlikar

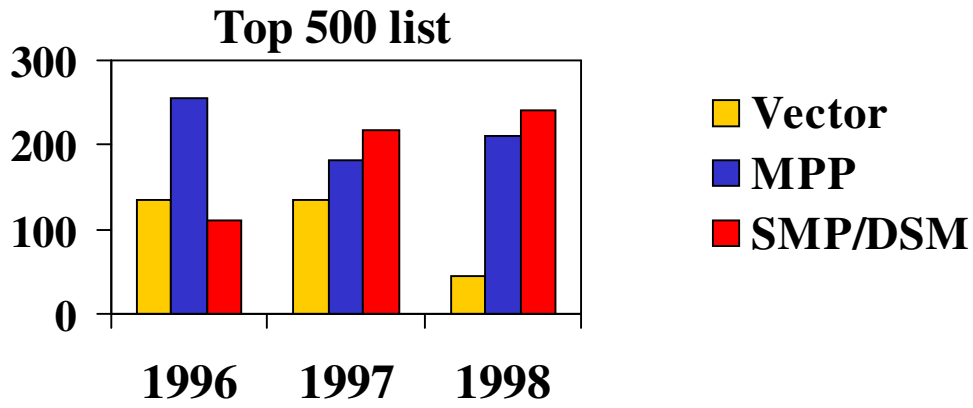
Joint work with Guy Blelloch

CMU Computer Science Dept.

http://www.cs.cmu.edu/~scandal/scheduling

Motivation

SMPs are everywhere...



We need a high level programming model to use them easily and effectively.

A solution: **lightweight, user-level threads.**

Summary

Lightweight Pthreads + space-efficient scheduler = simple code + good performance

Applications:

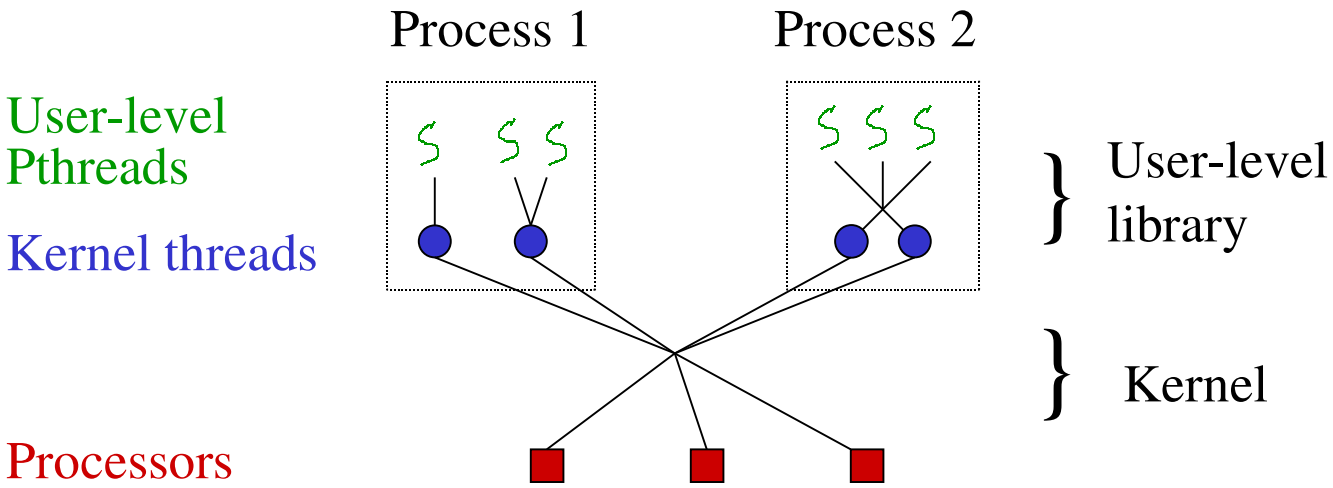
- Barnes Hut
- FMM
- Dense Matrix Multiply
- Sparse Matrix Multiply
- Decision Tree Builder
- FFTW
- Volume Rendering

Result: Up to 44% less running time, 63% less memory compared to original Pthreads scheduler.

Why Pthreads?

- POSIX standard threads, available on most platforms
  - Library approach (not a new language)
  - General threads functionality
  - User-level implementations (e.g., Solaris, Digital Unix, IRIX, AIX, HP-UX)

Two-level scheduling in Solaris:



Why Space Efficiency?

- Coarse-grained threads are difficult to use:
  - Requires explicit partitioning, scheduling and balancing of load
  - Cannot adapt to a changing number of processors

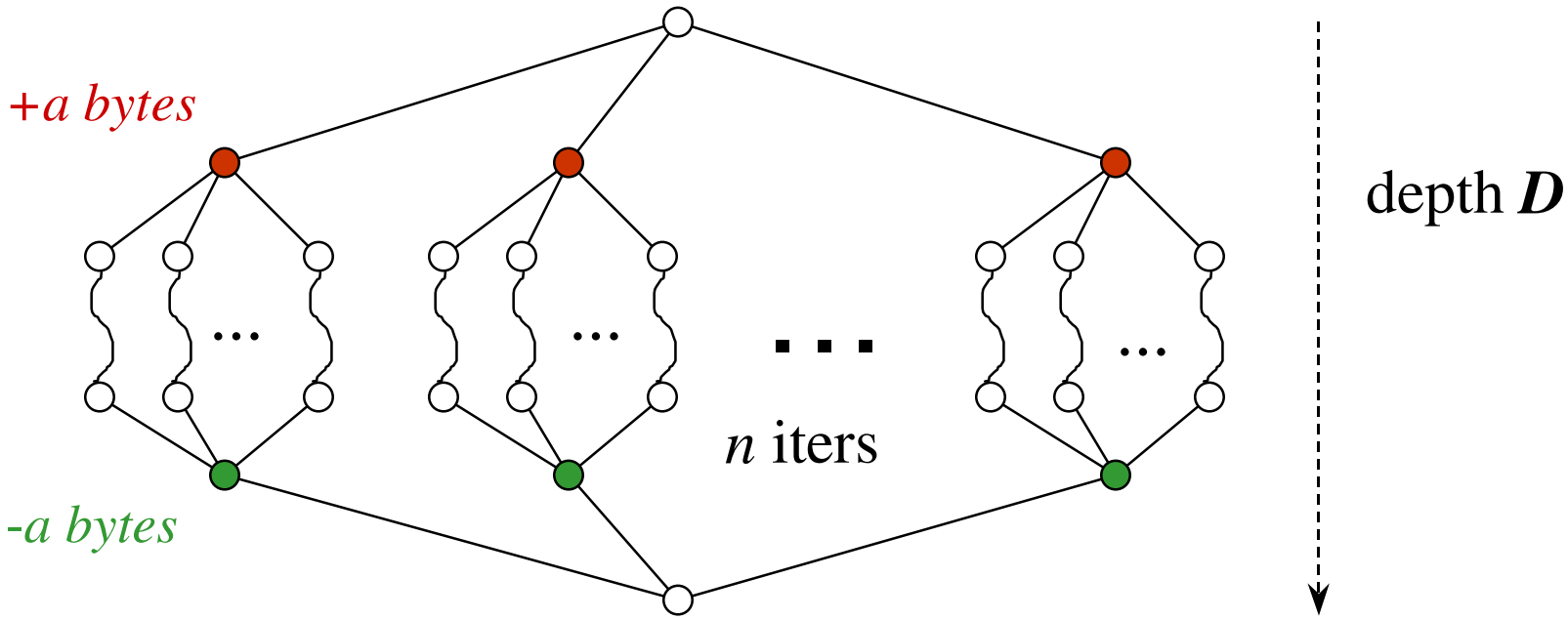
- Pthreads are still used in a coarse-grained style
  - Current Pthread schedulers not well suited to handle many threads
  - Too many active threads + excessive space allocation
    $\Rightarrow$ poor performance

We present a space-efficient scheduler for Pthreads.

Previous work on space-efficiency

- Heuristic-based approaches.
  - E.g., [Halstead 85], [Ruggiero-Sargeant 87], [Culler-Arvind 88].
- Provably-efficient techniques.
  - $S_p = p\, S_1$ [Burton-Simpson 94], [Blumofe-Leiserson 94],
  - $S_p = S_1 + O(pD)$ [Blelloch-Gibbons-Matias 95], **[Narlikar-Blelloch 97].**

Example

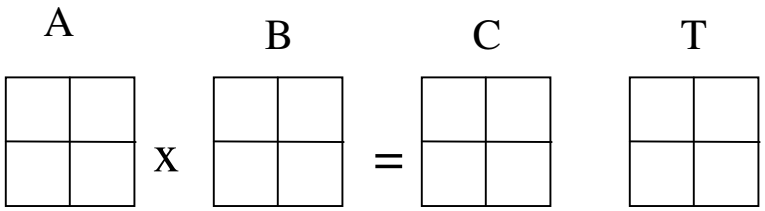


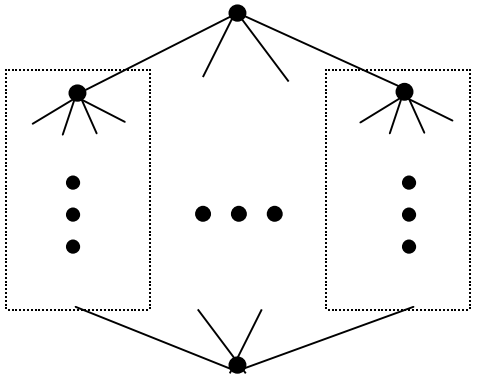
$S_1 = O(a)$

Previous approaches: $S_p = O(na)$ *or* $O(pa)$

Current approach: $S_p = O(a + pD)$

Divide-and-conquer Matrix Multiply

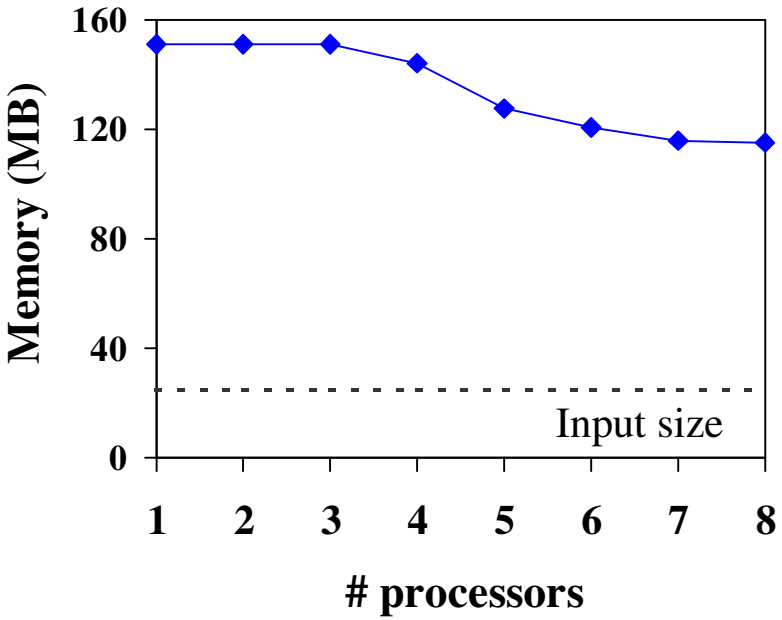


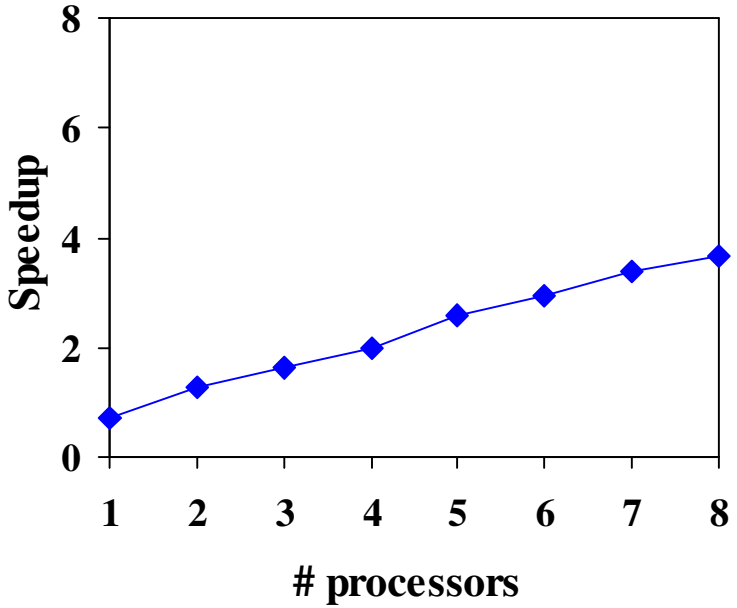


```
MM(A,B,C,size) {
 if (size < 64) SerMM(A,B,C,size);
 else
  T = allocate(size*size);
  fork MM(A11,B11,C11,size/2);
  ...
  fork MM(A22,B22,T22,size/2);
  synchronize with children;
  MatrixAdd(C, T, size);
  Free(T);
}
```

Time and Space Performance

Using the original Solaris 2.5 Pthreads library on Enterprise 5000 SMP





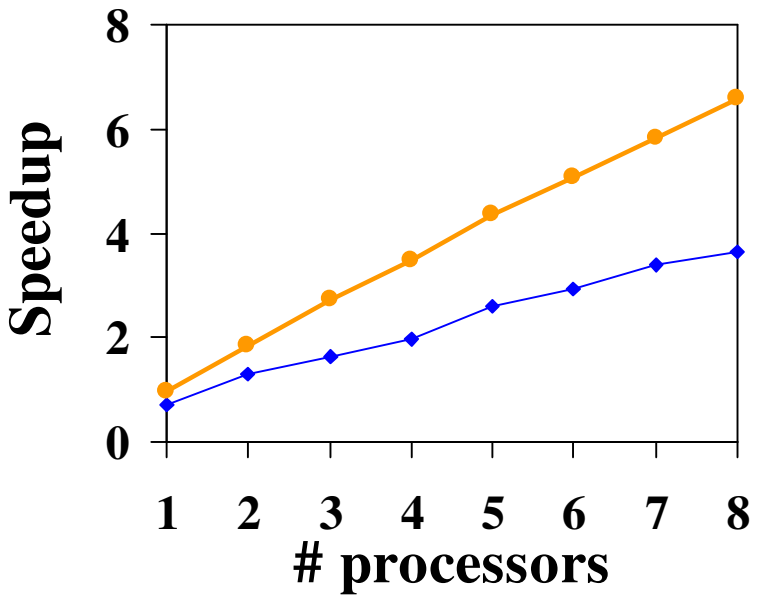
Conclusion: we need a space-efficient scheduler.

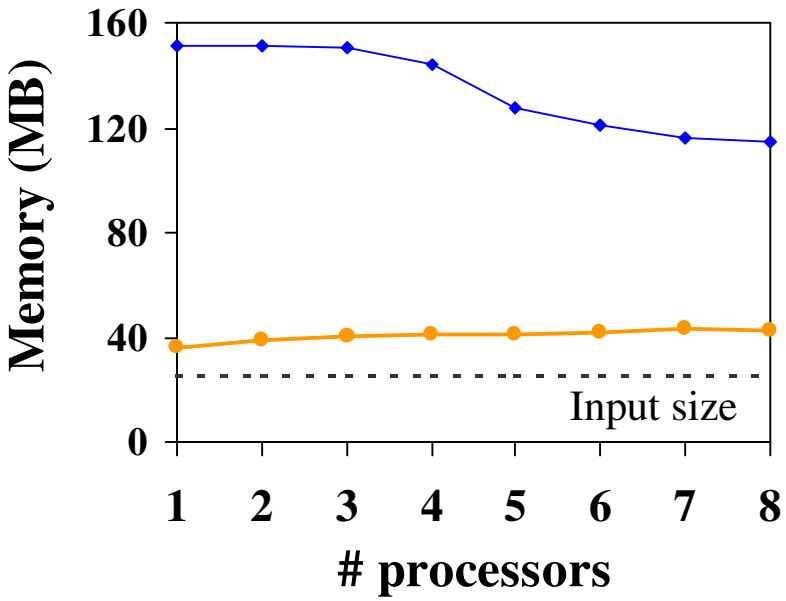
Our Space-Efficient Scheduler

- **Parallel depth-first order**. Schedule threads in an order as close as possible to the serial order.
- **Delayed allocations**. Delay big allocations to allow other threads to execute instead.
- **Minimal excess parallelism**. Don't create too many active threads.
- **Uninterrupted execution**. Keep a thread running on the same processor until it completes, forks or allocates too much space.

Space-Efficient Scheduler

Recursive Matrix Multiply





Original

Space-Eff+stk

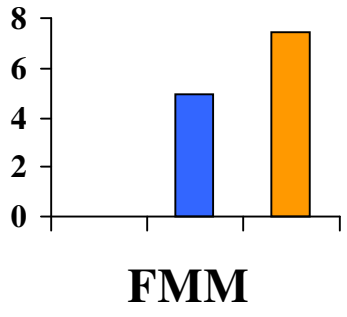
+ Smaller default stacksize (8KB instead of 1MB)

Other Parallel Benchmarks

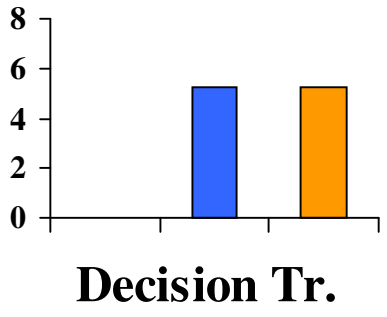
- Barnes Hut
  - Splash-2 benchmark suite
  - fine-grained version needs no costzones load balancing
- FMM
  - Uniform, fine-grained version
- Decision Tree Builder
  - C4.5-like algorithm for speech recognition (continuous attributes)
  - Divide-and-conquer; each stage has a divide-and-conquer qsort
- FFTW
- Sparse Matrix-Vector Multiply
  - Earthquake data
- Volume Rendering
  - Splash-2 benchmark suite

Speedups for Parallel Benchmarks

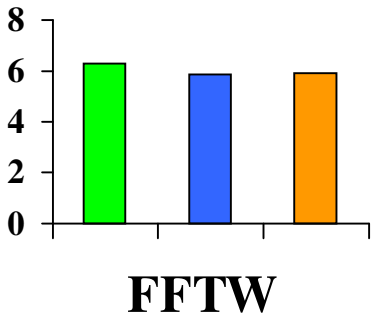
Barnes Hut



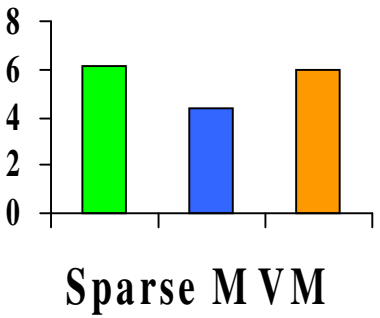
FMM



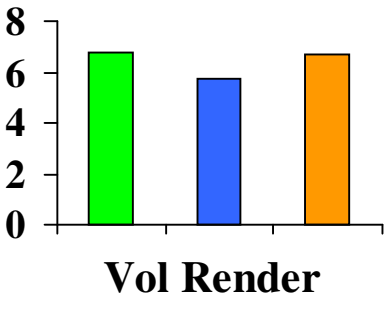
Decision Tr.



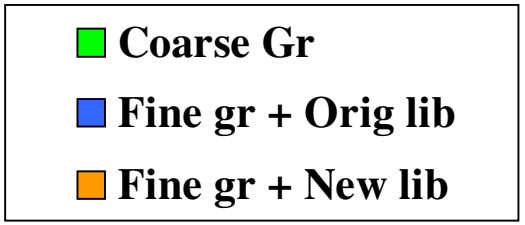
FFTW



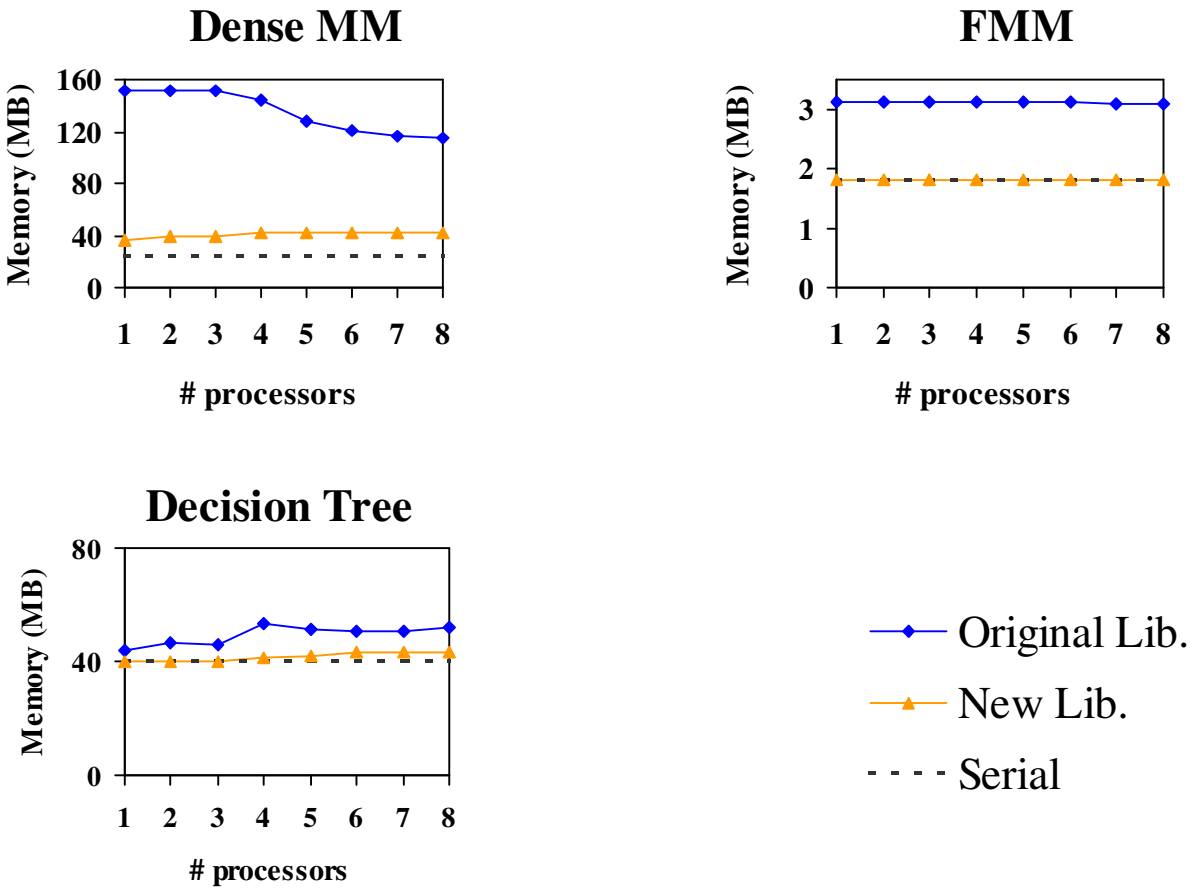
Sparse M VM



Vol Render



Dynamic Memory Requirement



Conclusions

- Our space-efficient scheduler can be implemented for Pthreads without affecting other Pthreads functionality.
- Lightweight Pthreads are a simple and efficient model for programs with dynamic and irregular parallelism, provided the right scheduler is used.
- Programs with fine-grained threads may even outperform their coarse-grained, hand-partitioned counterparts.
- For threads library implementors: important to separate out the scheduler to allow modifications and extensions.