Lessons Learned from the Shared Memory Parallelization of a Functional Array Language

Clemens Grelck
University of Lübeck, Germany

Outline of Talk:

- Functional array programming with SAC.
- Choosing shared memory systems.
- Organization of parallel program execution.
- Architecture-specific pitfalls.
- Conclusion: lessons learned.

Functional Array Programming in SAC

Characteristics:

- Array: multidimensional abstract data structure.
- Array: data vector + shape vector.
- Creation / projection facilities.
- Call-by-value parameter passing.
- Memory management by compiler / runtime system.

Example:

```
bool continue( double[+] A, double[+] A_old, double eps)
{
  return( any( abs( A - A_old) >= eps));
}
```

The With-Loop Construct

Example:

```
bool[+] >= ( double[+] A, double b)
{
  res = with (. <= i_vec <= .) : A[i_vec] >= b ;
        genarray( shape( A));

  return( res);
}
```

In general:

```
  res = with  index_set_1  :  expr_1  ;
                  ...          ...
              index_set_n  :  expr_n  ;
        genarray(  shp_expr  );
```

Parallelization for Shared Memory

What everyone does:

* Message passing / MPI

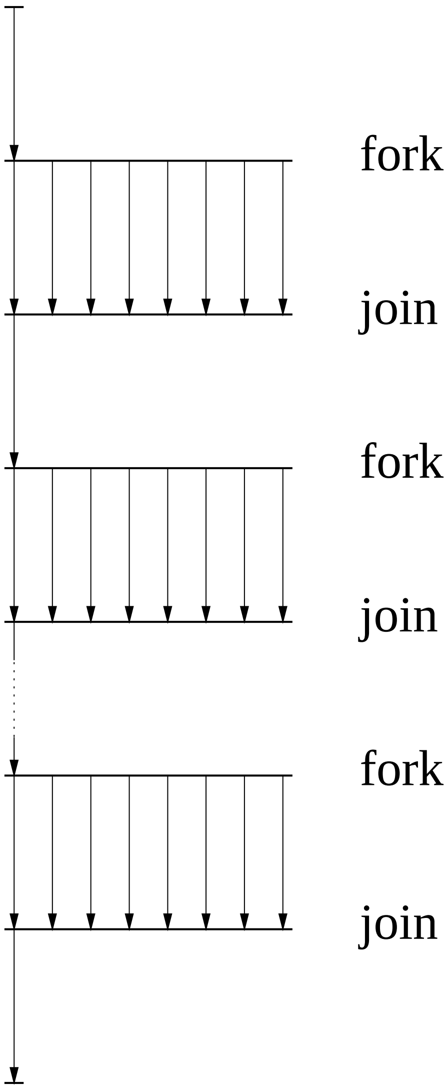
What we do:

* Multithreading / PThreads

Pragmatics:

* No explicit data decomposition:
  $\Longrightarrow$ adopt sequential memory data layout.
* Only array operations affected:
  $\Longrightarrow$ sequential code for I/O, etc. remains as is.
  $\Longrightarrow$ focus on compilation of with-loops.
  $\Longrightarrow$ partly reuse existing sequential compilation scheme.

Multithreaded Program Execution



fork

join

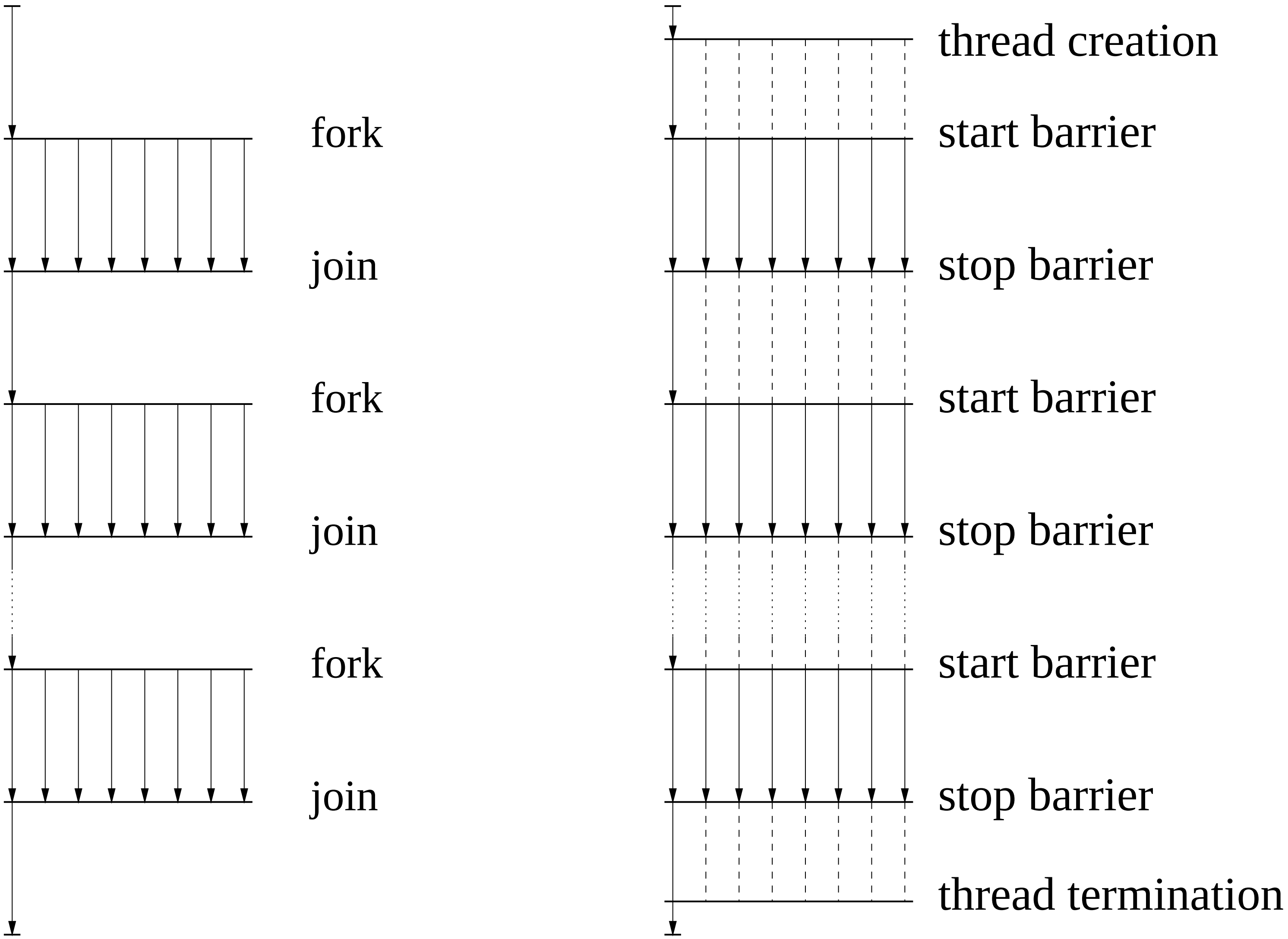
fork

join

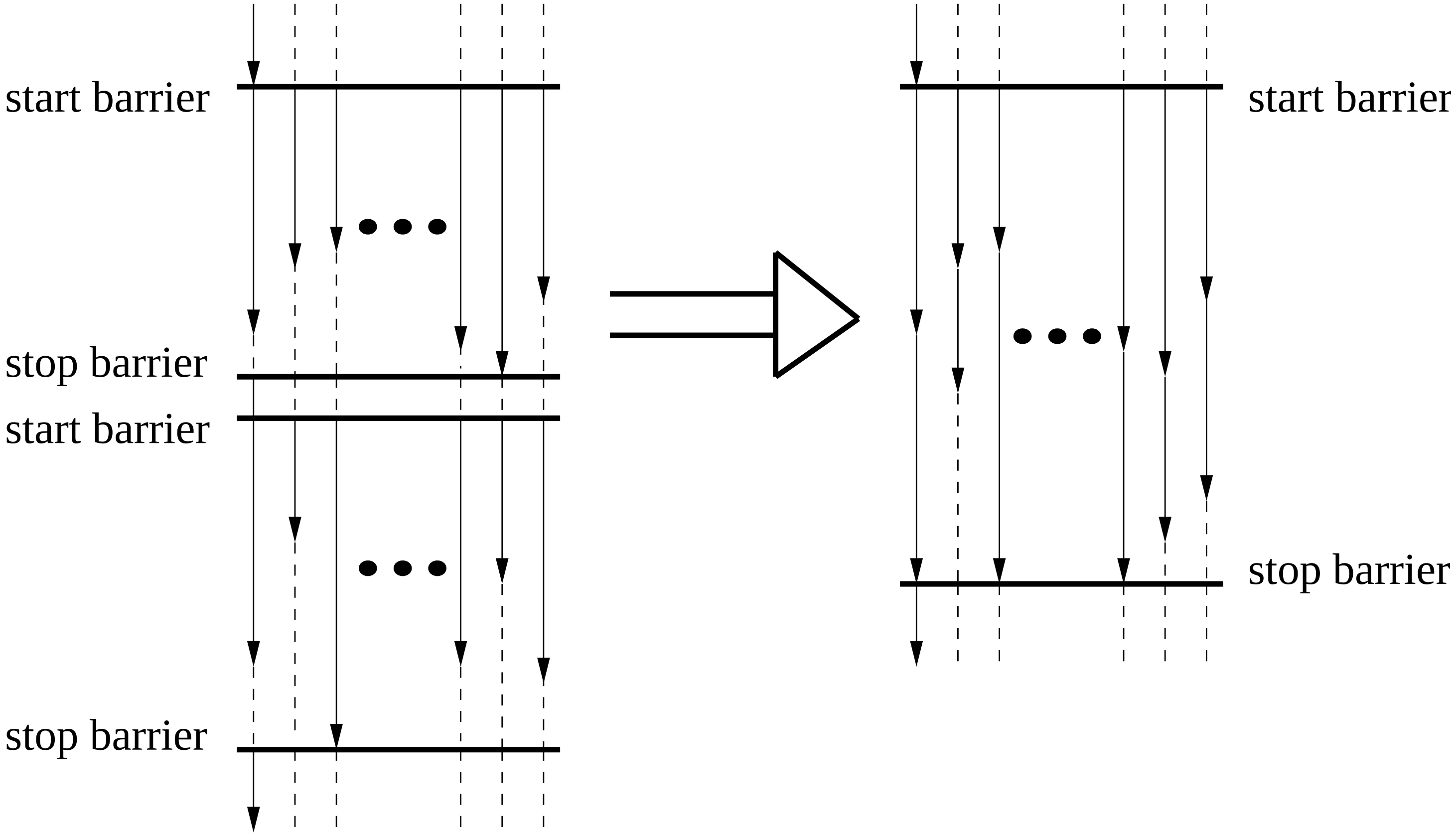
fork

join

Multithreaded Program Execution

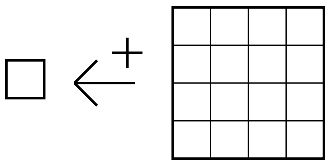


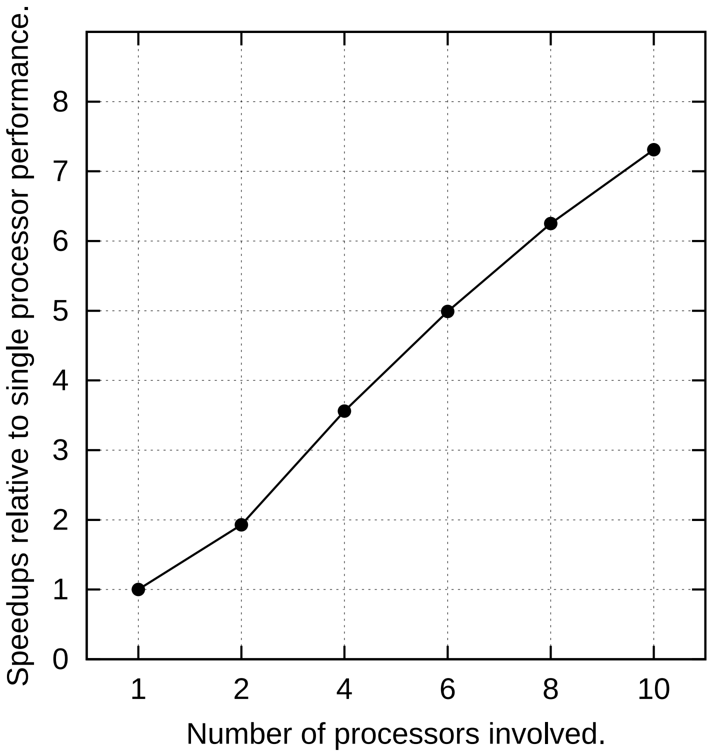
Avoiding Synchronization Barriers



Experimental Evaluation

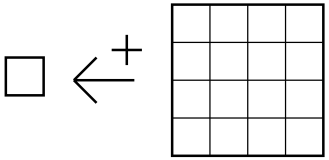
Sum 1:

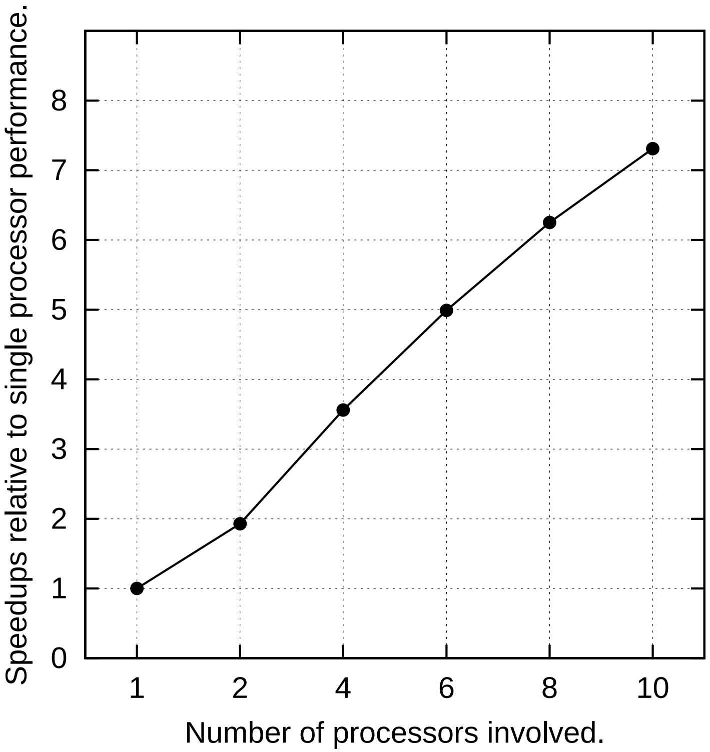




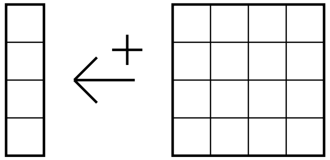
Experimental Evaluation

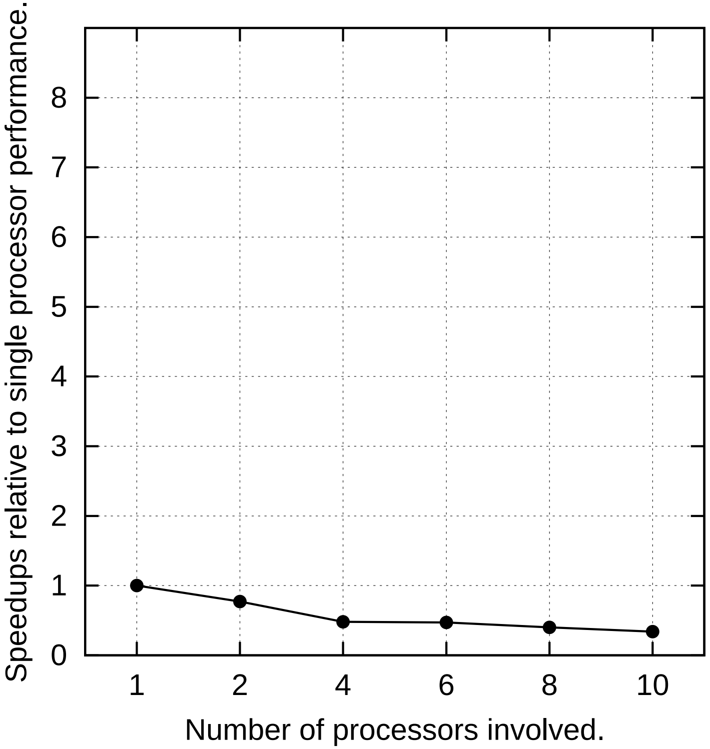
Sum 1:





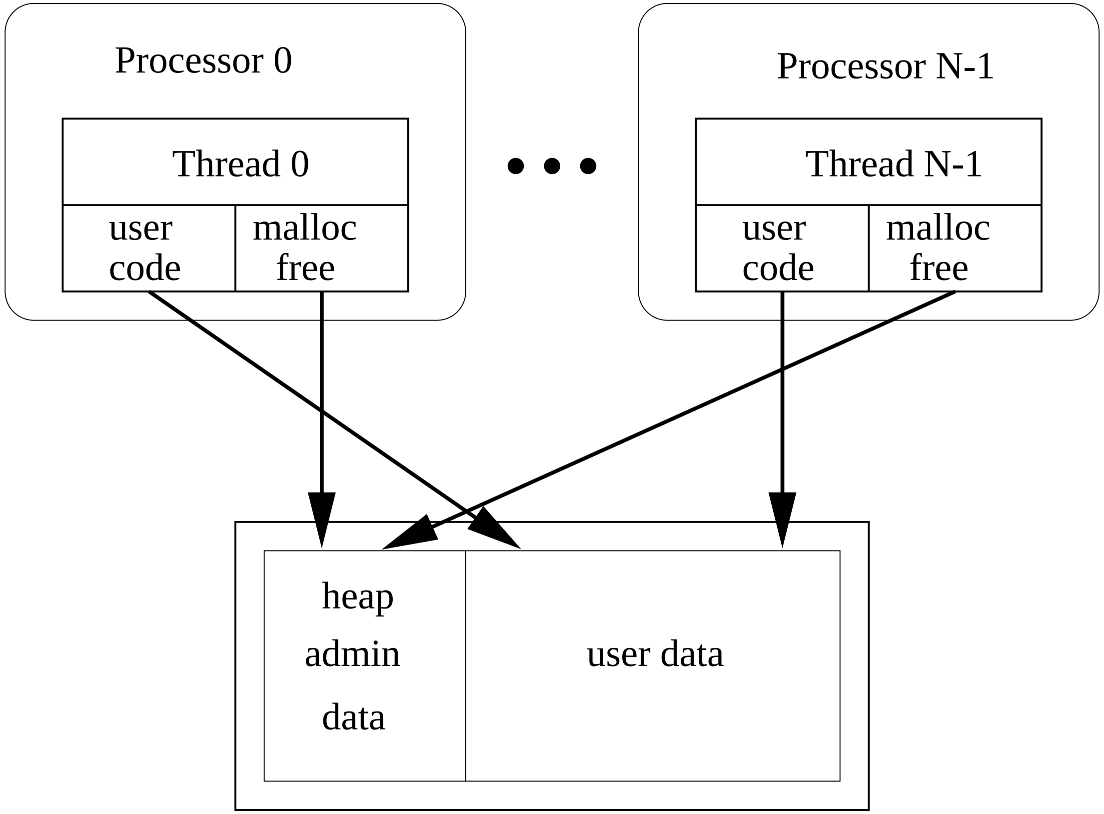
Sum 2:





Multithreaded Memory Management

Problem Identification:



Processor 0

Thread 0

user code

malloc free

• • •

Processor N-1

Thread N-1

user code

malloc free

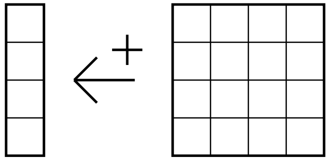
heap admin data

user data

Heap Memory

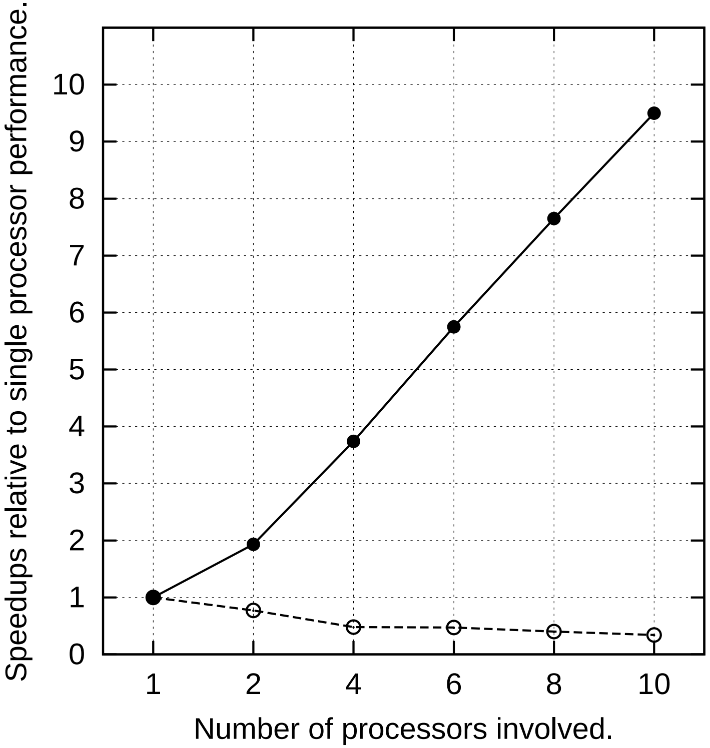
Private Memory Manager

Sum 2:



Organization:

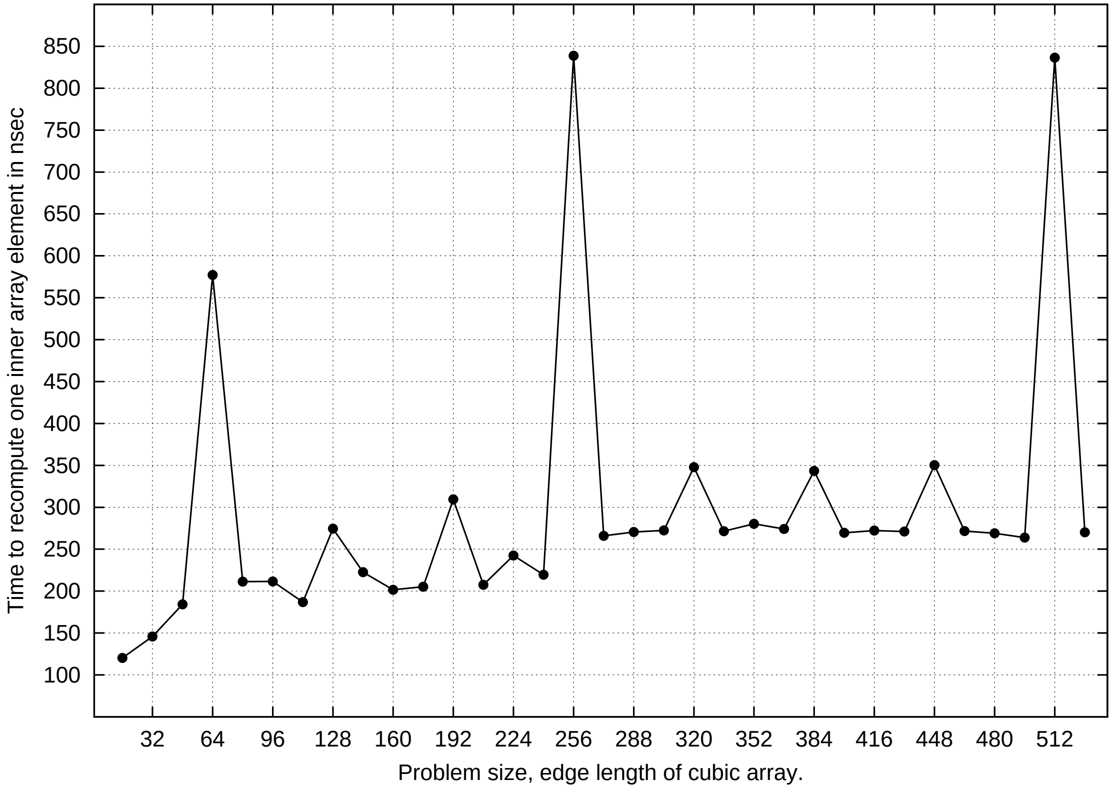
- Hierarchy of nested heaps.
- Private subheaps for individual threads.
- Tight integration into runtime system.
- Exploitation of compile time knowledge.
- Exploitation of runtime knowledge.



Performance Impact of Cache Memories

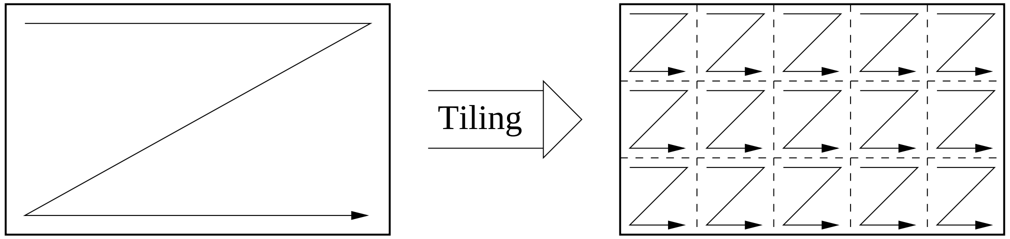
- 3-dimensional relaxation kernel.
- Systematic variation of grid size: 16^3 (32KB) $\longrightarrow$ 528^3 (1.2GB)

Time to update single grid point:

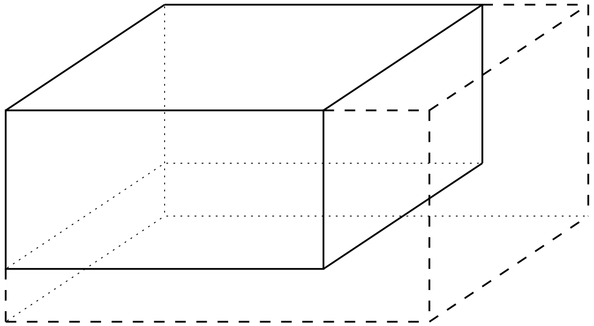


Cache Optimizations

With-Loop Tiling:



Array Padding:

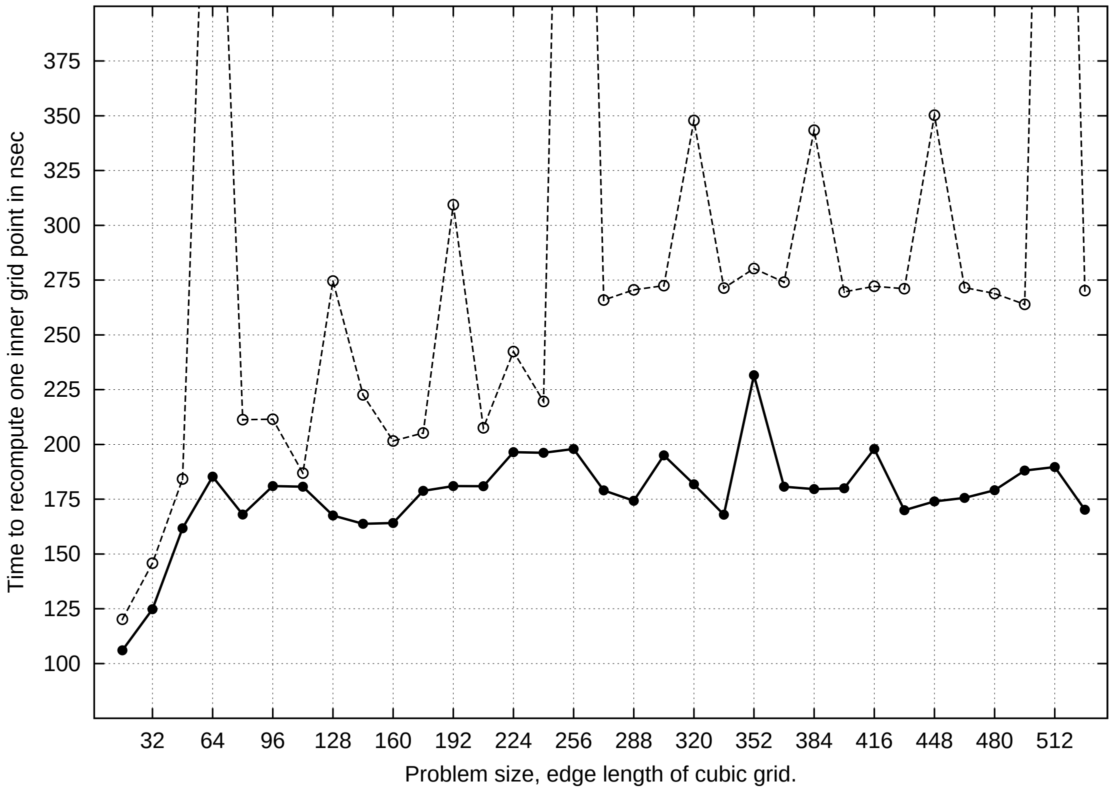


Array Placement:

Performance Impact of Cache Optimizations

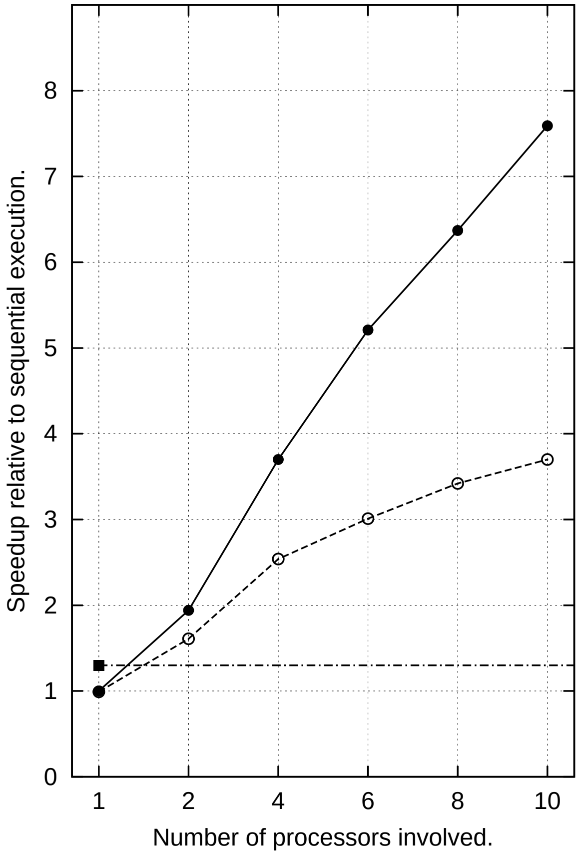
- **Padding:** 25 out of 33 problem sizes
- **Tiling:** 19 out of 33 problem sizes
- **Placement:** always

Time to update single grid point:

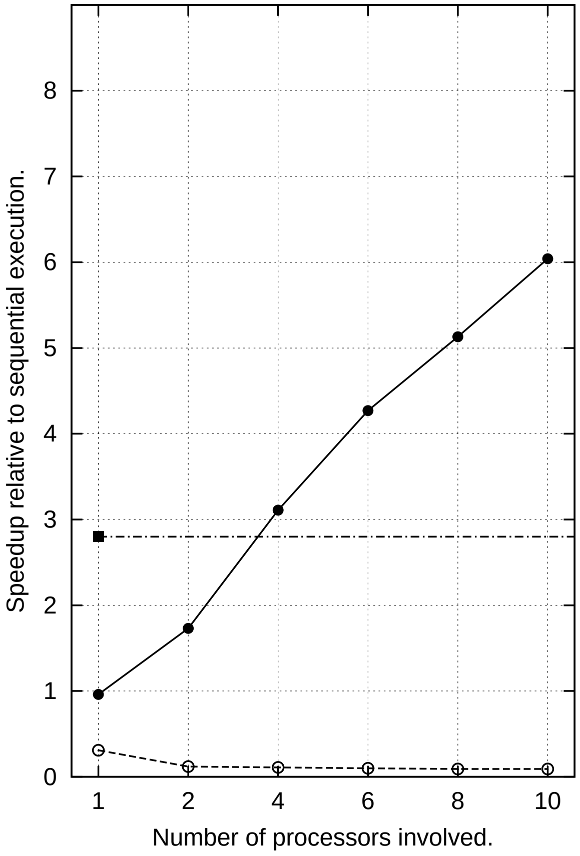


Experimental Evaluation

NAS Benchmark MG:



NAS Benchmark FT:



Conclusion

Fairly simple:

*** Non-sequential program execution**

⟹ Functional approach pays off.

⟹ Shared memory architecture pays off.

Fairly difficult:

*** Achieving desired speedups**

⟹ Fine-tuned runtime system.

⟹ Tailor-made dynamic memory management.

⟹ Various cache optimizations.