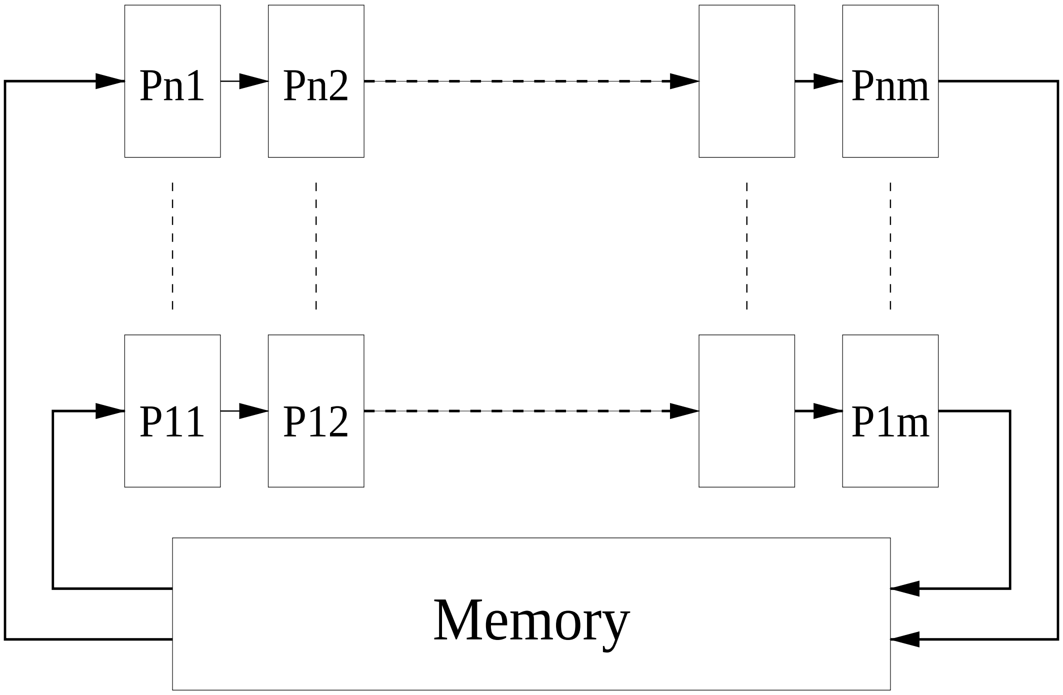
Performance Aspects of Iterative Methods on Superscalar Computers

Ulrich Rüde

Institut für Mathematik
Universität Augsburg



Ideally ...

we can use

- multigrid with nested iteration to solve scalar elliptic PDE with approximately
  - 100 operations
  - storing 8 reals

  per unknown
- on a workstation
  - that can do 800×10^6 operations (= 800 MFlop) per second
  - in 64×10^6 words (=512 MByte) of storage

and so, we can solve for

8×10^6 unknowns in about $\boxed{\text{one}}$ second

.

In practice ...

we are happy when our algorithm

- can do 10^5 unknowns
- on a parallel system
- where it doesn't run for hours

Where the performance is lost

The ideal picture may be overly optimistic since the problem may be more complicated, like

- non-symmetric
- non-linear
- systems

needing more expensive algorithms.

On the other hand it may be that we use

- non optimal algorithms
- non optimal implementations (for a given machine architecture).

This talk will try to shed some light on the implementation aspect.

CPU (R)Evolution

- Workstations and
- parallel computers

both increasingly emply standard microprocessors. They have

- pipelined execution with 5-10 stages.
- superscalar execution with 2 (or 4) floating point pipelines
- clock rates of up to 400 (soon 500) MHz
- peak MFlop of 800 (soon 1 GFlop)

The performance increase is approx. 80 % per year.

Thus any modern CPU is a $\boxed{\text{Parallel Processor}}$ in itself, employing typically something like 20-40 parallel units (and more in the future).

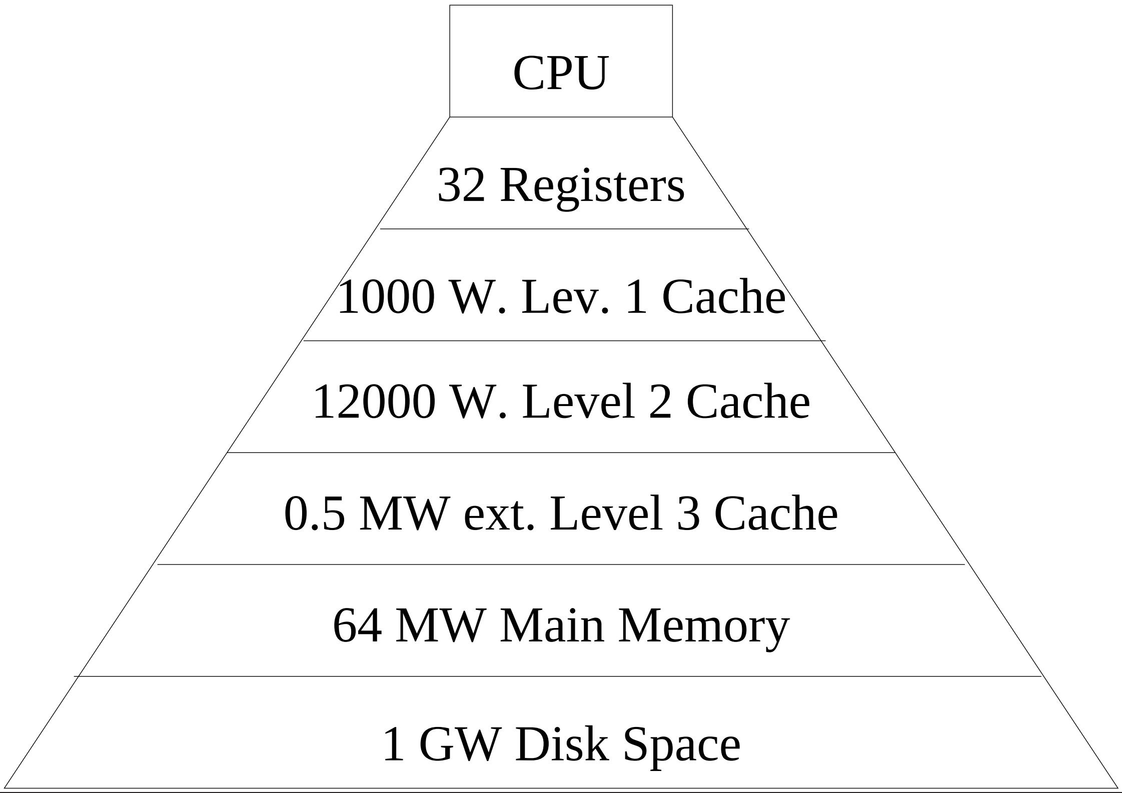
Memory

- DRAM still has access times of only 60 ns.
- access times decrease, but only at about 5% per annum.
- additional overhead for memory access on the order of 100ns.
- virtual shared memory parallel processors are an additional order of magnitude slower.

Thus all systems have

- memory speed $\ll$ processor speed
- a hierarchical memory system with up to 3 levels of caches with successively longer latencies and lower bandwith

Memory Hierarchy for DEC AlphaServer 8400



Level	Capac.	Thruput	Latency
Register	512 B	34 GB/s	3 ns
Cache 1	8 KB	9.6 GB/s	3 ns
Cache 2	96 KB	4.8 GB/s	10 ns
Cache 3	4 MB	1 GB/s	20 ns
Main Mem	14 GB	300 MB/s	200 ns

Dot product (BLAS `ddot`)

```
sum= 0.0d0
do i=1,n
   sum= sum+ a(i)*b(i)
enddo
```

HP755: ca: 90 MFlop(maximal)

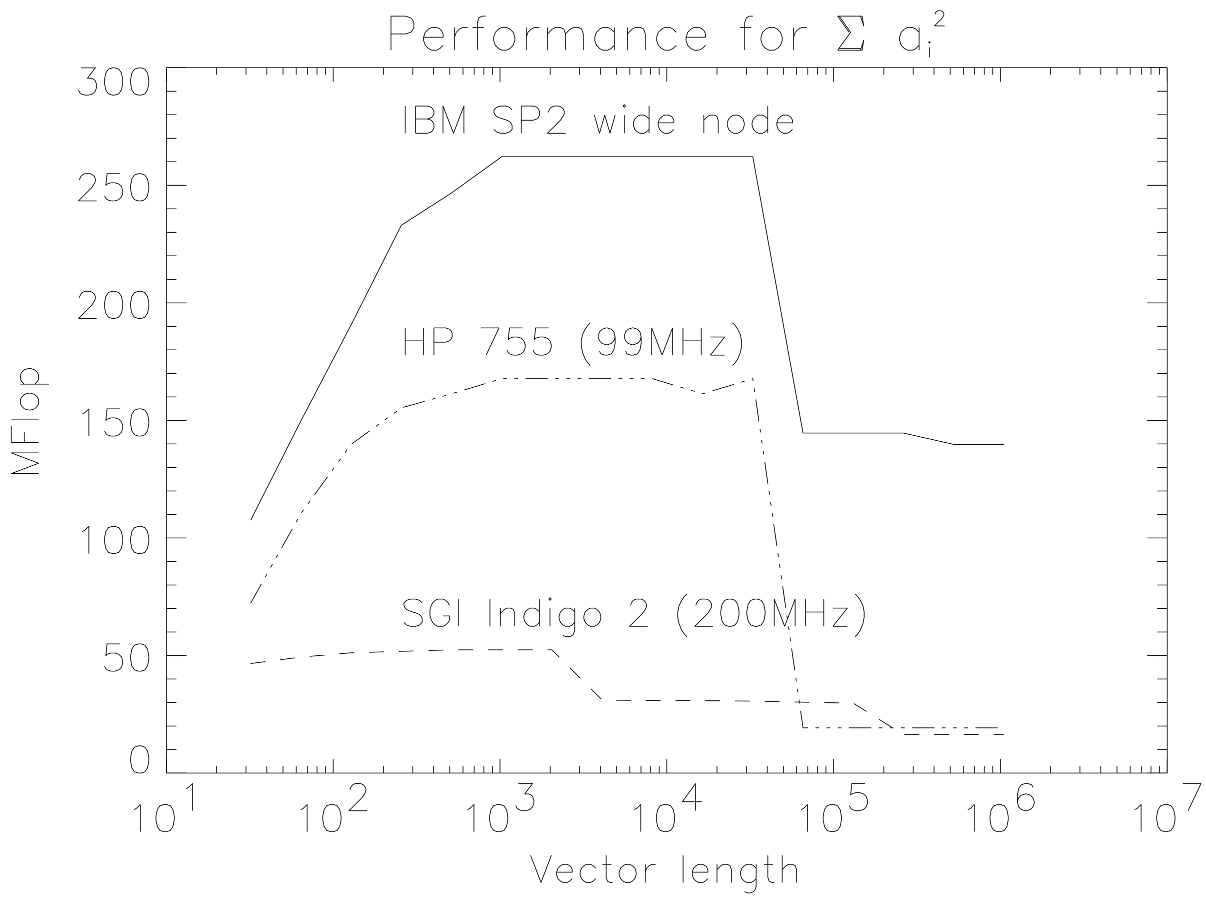
Unrolled dot product

```
sum1= 0.0d0
sum2= 0.0d0
sum3= 0.0d0
sum4= 0.0d0
do i=1,n,4
   sum1= sum1+ a(i)*b(i)
   sum2= sum2+ a(i+1)*b(i+1)
   sum3= sum3+ a(i+2)*b(i+2)
   sum4= sum4+ a(i+3)*b(i+3)
enddo
sum= sum1+sum2+sum3+sum4
```

HP755: ca: 160 MFlop(maximal)

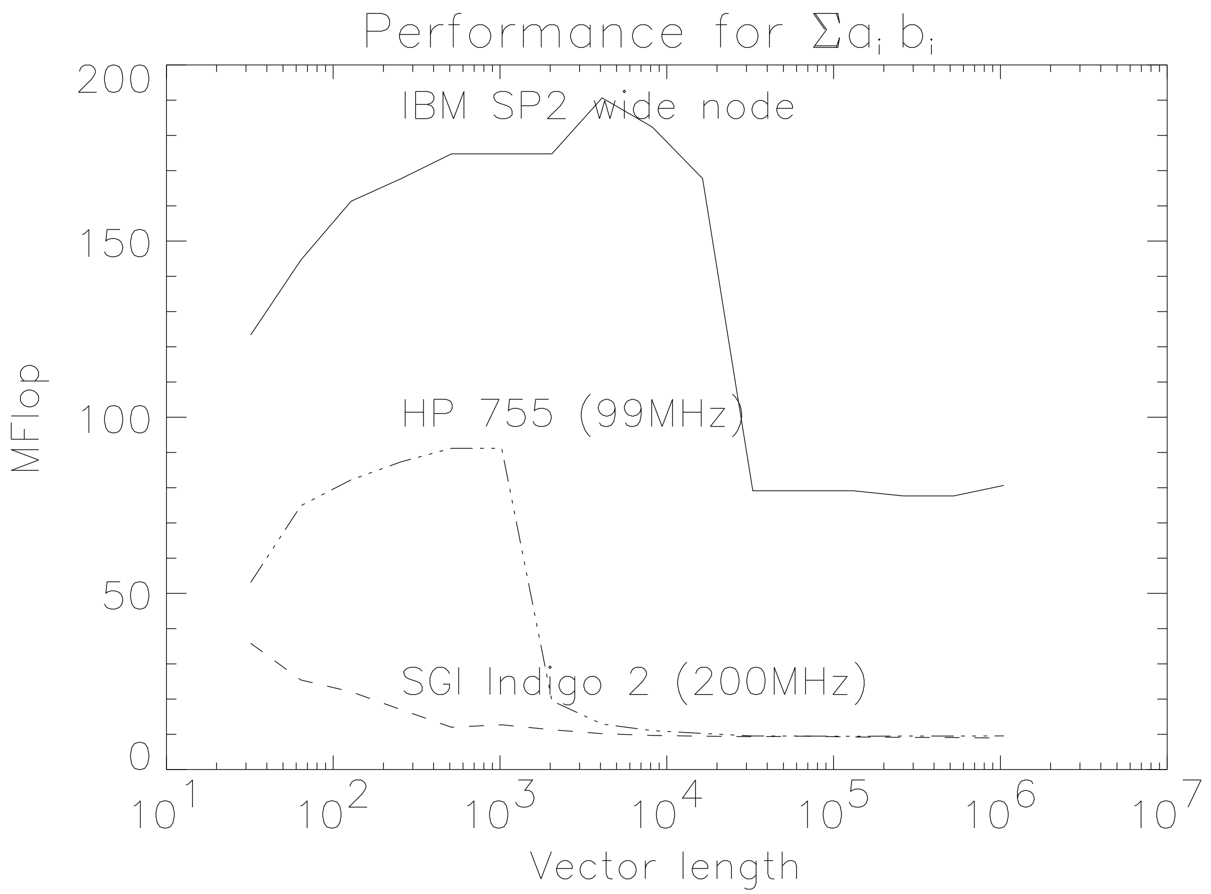
Performance depending on vector length

- $\sum_1^n a_i^2$ for $n = 32, 64, \ldots, 1048576$
- a $\equiv$ b
- Stride 1
- One read per Multiply-Add pair



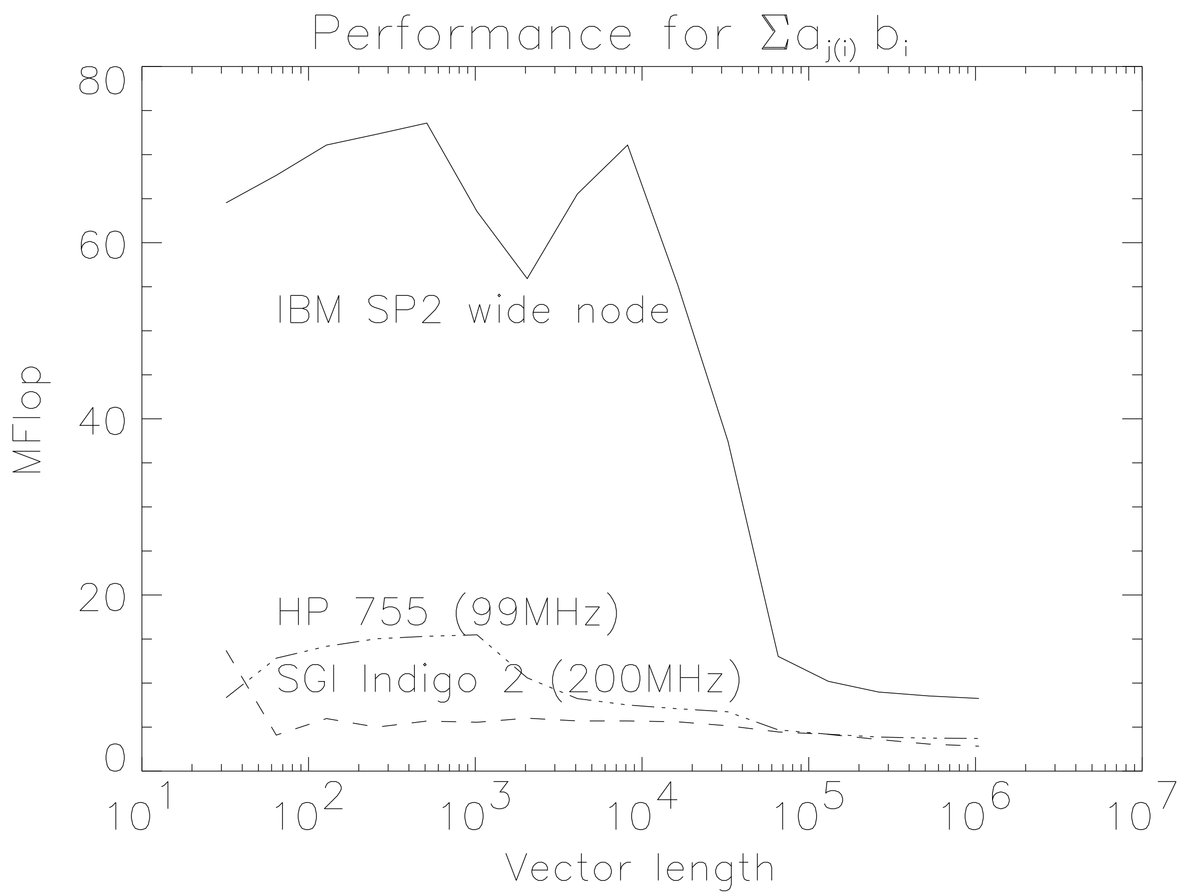
Performance depending on vector length

- $\sum_{1}^{n} a_i b_i$ for $n = 32, 64, \ldots, 1048576$
- a $\not\equiv$ b
- Stride 1
- two data sets, possible cache associativity conflicts
- two reads per Multiply-Add pair



Performance depending on vector length

- $\sum_{1}^{n} a_{j_i} b_i$ for $n = 32, 64, \ldots, 1048576$
- $\mathtt{a} \not\equiv \mathtt{b}$
- Stride 1 for `b`
- Indirect addressing for `a`
- two reads per Multiply-Add pair (plus index array)



What does that mean?

What determines the run time of an algorithm?

− Processor (# operations)

+ Memory (data layout)

  + *temporal* locality of memory accesses

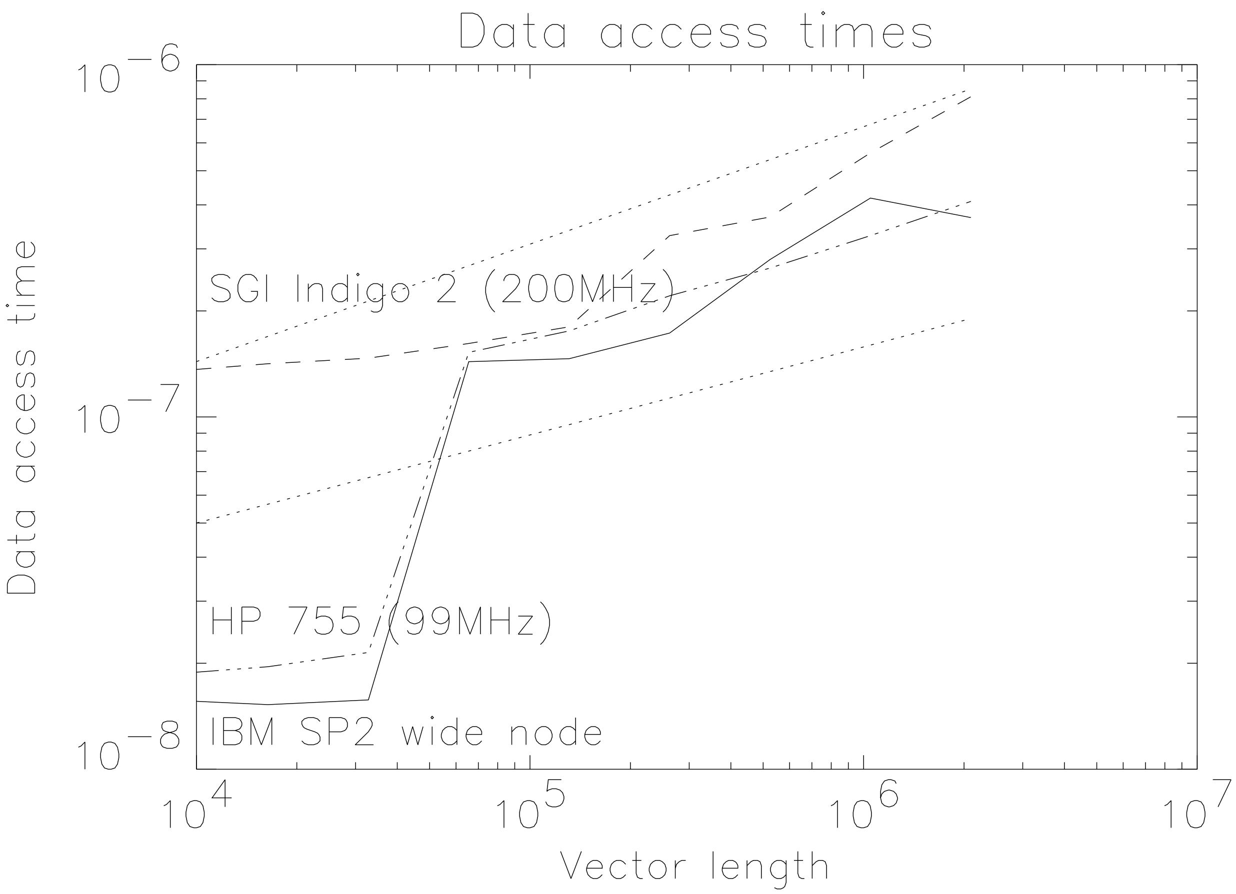
  + *spatial* locality of memory accesses

+ use of CPU-internal parallelism

We need performance models which help to *evaluate* the realistic performance of algorithms and *design* the algorithms in the correct way in the future.

Problem: Real behavior of modern computers is difficult to *understand* and describe analytically.

Data Access Cost



The measurements suggest that the access cost is a function of data set size,

$$C(n) \approx \begin{cases} O(n^{(1/3)}) \\ O(n^{(1/4)}) \end{cases}$$

Memory Access Performance Model

Simplified assumption: The cost to access address a is a function of the time, when a has been accessed last. So, when a program accesses a sequence of addresses

$$a_1, a_2, a_3, \ldots$$

at times $i = 1, 2, \ldots,$ the cost for the access at time k is

$$\tilde{C}(a_k) = C(L(a_k)),$$

where the *last access* $L(a_k)$ is defined as

$$L(a_k) = \begin{cases} \min\limits_{a_j = a_k, j<k} k - j & \text{if defined} \\ L_{\text{max}} & \text{otherwise} \end{cases}$$

and where $C(n) \approx n^\alpha$, $\alpha = 1/3, 1/4$.

Memory Performance Analysis for Relaxation Methods (1)

```
do it=1,N
do i=1,N-1
do j=1,N-1
   y= 0.25 * (u(j,i-1)+u(j-1,i)+
             u(j,i+1)+u(j+1,i))
   x= y - u(j,i)
   u(j,i)= u (j,i) + w*x
enddo
enddo
enddo
```

Cost in each pass (except the first)

Datum x	$L(x)$	$C(L(x))$
`u(j,i+1)`	$O(N^2)$	$O(N^{2\alpha})$
`u(j+1,i)`	$O(N)$	$O(N^\alpha)$
`u(j,i)`	$O(1)$	$O(1)$
`u(j,i-1)`	$O(1)$	$O(1)$
`u(j-1,i)`	$O(N)$	$O(N^\alpha)$

Memory Performance Analysis for Relaxation Methods (2)

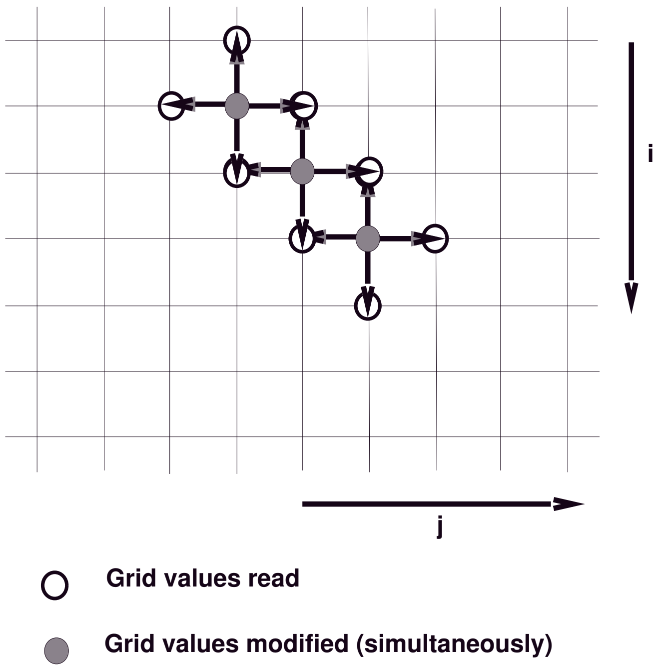
Cost for one full sweep:

$$O(N^2 * (C(N^2) + 2C(N) + 2C(1)) = O(N^{2+2\alpha})$$

With over-relaxation $O(N)$ sweeps are necessary, so the cost of the solution is

$$O(N) * O(N^{2+2\alpha}) = O(N^{3+2\alpha})$$

Pipelined Iterations (Fusing iterations)



Cost for k iterations in one pipeline

$$N^2(C(N^2) + (k+1)C(N) + 2kC(1))$$

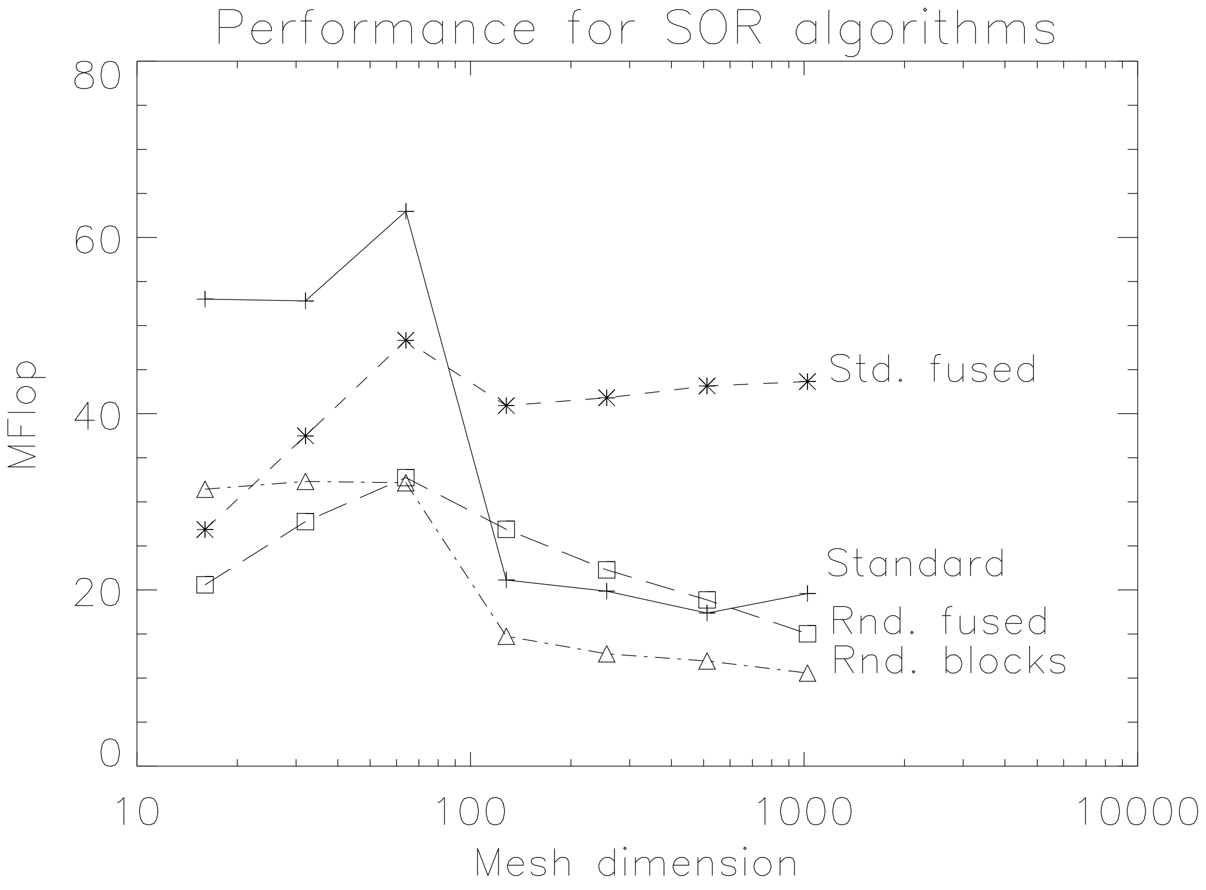
Cost for N/k such sweeps performing k iterations:

$$O\left(\frac{N^3}{k}N^{2\alpha} + N^3\frac{k+1}{k}N^{\alpha} + 2N^3\right)$$

With $k = N^{\alpha}$, the total cost is $O(N^{3+\alpha})$.

Relaxation Methods: Performancs Evaluation

SOR for $n = 16, 32, \ldots, 1024$



Comparison of iterative algorithms

- Standard Multigrid
- Adaptive Multigrid
- SOR
- SOR with cache optimization