Improving Parallelism and Locality with Asynchronous Algorithms

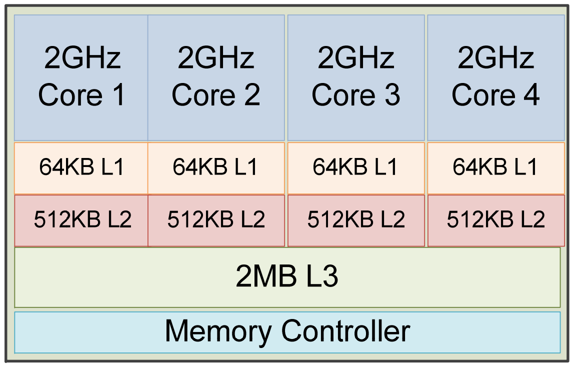
Lixia Liu, Zhiyuan Li
Purdue University, USA
PPOPP 2010, January 2009

Work supported in part by NSF through grants ST-HEC-0444285, CPA-0702245 and CPA-0811587, and by a Google Fellowship

Background

- Multicore architecture
  - Multiple cores per chip
  - Modest on-chip caches
  - Memory bandwidth issue
    - Increasing gap between CPU speed and off-chip memory bandwidth
    - Increasing bandwidth consumption by aggressive hardware prefetching

AMD 8350



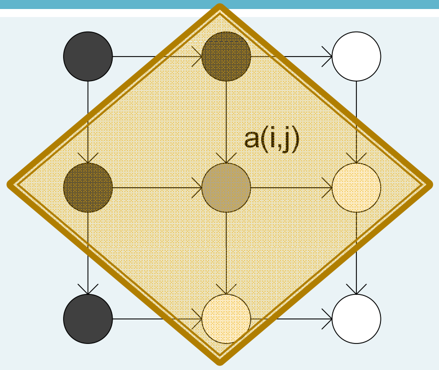
- Software
  - Many optimizations increase memory bandwidth requirement
    - Parallelization, Software prefetching, ILP
  - Some optimizations reduce memory bandwidth requirement
    - Array contraction, index compression
  - Loop transformations to improve data locality
    - Loop tiling, loop fusion and others
    - **Restricted by data/control dependences**

Background

- Loop tiling is used to increase data locality
- Example program: PDE iterative solver

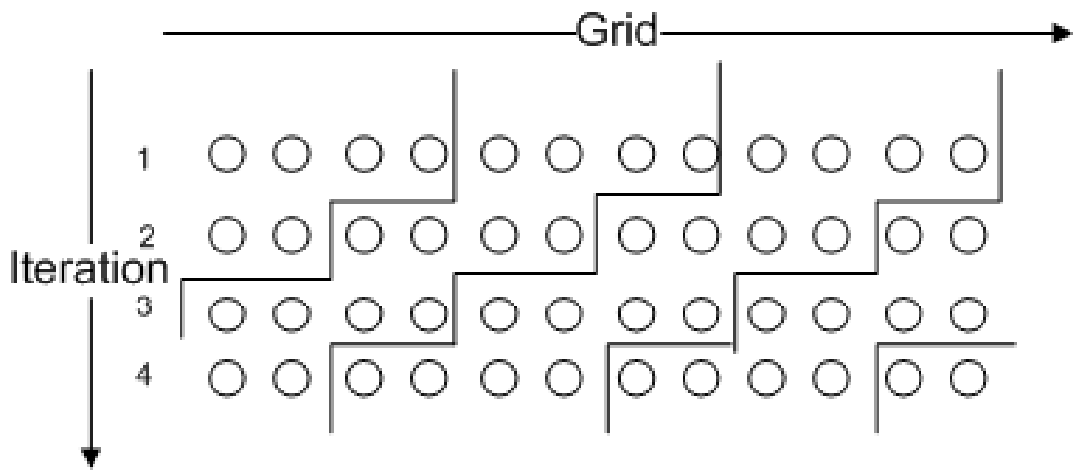
The base implementation

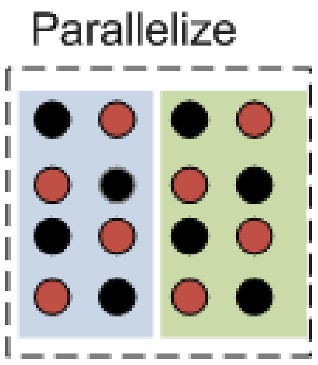
```
do t = 1,itmax

  update(a, n, f);

  ! Compute residual and convergence test
  error = residual(a, n)

  if (error .le. tol) then
    exit
  endif
end do
```



Loop tiling

- Tiling is skewed to satisfy data dependences
- After tiling, parallelism only exists within a tile due to data dependences between tiles





Loop tiling code

The tiled version with speculated execution

```
do t = 1, itmax/M + 1

  ! Save the old result into buffer as checkpoint
  oldbuf(1:n, 1:n) = a(1:n, 1:n)

  ! Execute a chunk of M iterations after tiling
  update_tile(a, n, f, M)

  ! Compute residual and perform convergence test
  error = residual(a, n)

  if (error .le. tol) then
    call recovery(oldbuf, a, n, f)
    exit
  end if
end do
```

Questions

1. How to select chunk size?
2. Is recovery overhead necessary?

Motivations

- Mitigate the memory bandwidth problem
  - Apply data locality optimizations to challenging cases
  - Relax restrictions imposed by data/control dependences

Asynchronous

- Basic idea: allow to use of old neighboring values in the computation, still converging
- Originally proposed to reduce communication cost and synchronization overhead
- Convergence rate of asynchronous algorithms[1]
  - May slowdown convergence rate
- Our contribution is to use the asynchronous model to improve parallelism and locality simultaneously
  - Relax dependencies
  - Monotone exit condition

[1] Frommer, A. and Szyld, D. B. 1994. Asynchronous two-stage iterative methods. In *Numer. Math.* 69, 2, Dec 1994.

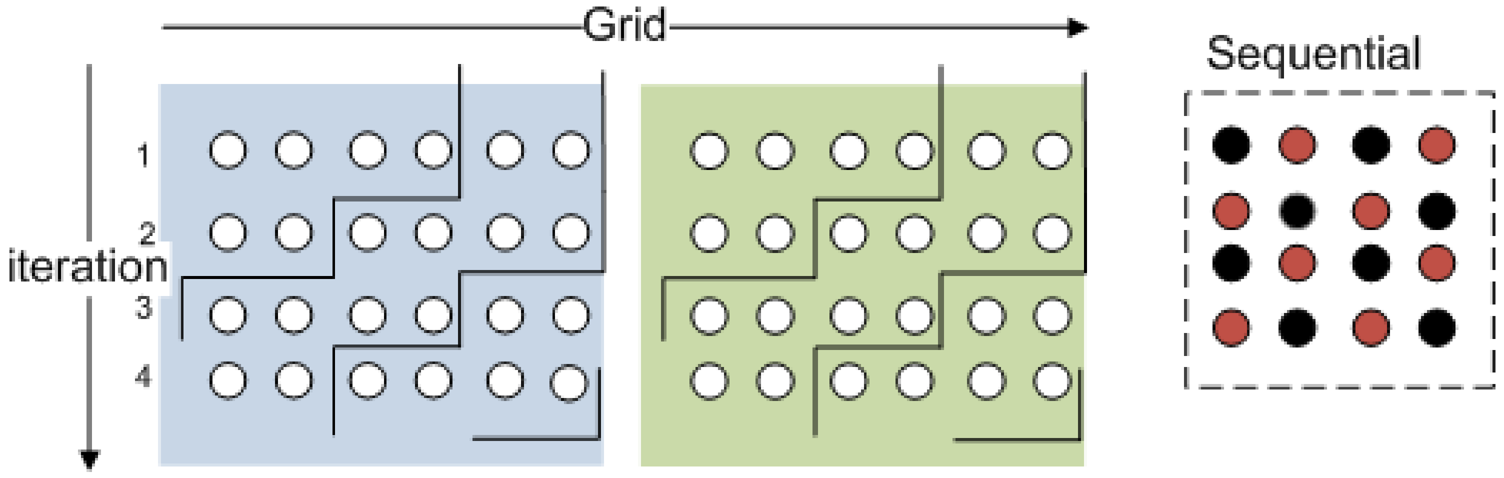
Tiling without recovery

The tiled version without recovery

```
do t = 1, itmax/M + 1

  ! Execute a chunk of M iterations after tiling
  update_tile(a, n, f, M)

  ! Compute residual and convergence test
  error = residual(a, n)

  if (error .le. tol) then
    exit
  end if
end do
```

Asynchronous model

- Achieve parallelism across the grid
  - Not just within a tile
- Apply loop tiling to improve data locality
  - Requiring a partition of time steps in chunks
- Eliminate recovery overhead



Chunk size selection

- *Chunk size*: # iterations executed speculatively in the tiled code
- Ideal if we can predict the exact iterations to converge
  - However, it is unknown until convergence happens
- Too large a chunk, we pay overshooting overhead
- Too small, poor data reuse and poor data locality

How to determine chunk size?

- Poor solutions
  - Use a constant chunk size (randomly pick)
  - Estimate based on the theoretical convergence rate

- A better solution: Adaptive chunk size
  - Use the latest convergence progress to predict how many more iterations are required to converge

$$C^* = \log\left(\frac{\text{tol}}{r_k}\right) \times C \Big/ \log\left(\frac{r_k}{r_{k-1}}\right) \quad \text{then} \quad C^* \to C$$

r_i :residual error of *i*-th round of tiled code

Evaluations

- Platforms for experiments:
  - Intel Q6600, AMD8350 Barcelona, Intel E5530 Nehalem
- Evaluated numerical methods: Jacobi, GS, SOR
- Performance results
  - Synchronous model vs. asynchronous model with the best chuck size
  - Original code vs. loop tiling
  - Impact of the chunk size
  - Adaptive chunk selection vs. the ideal chunk size

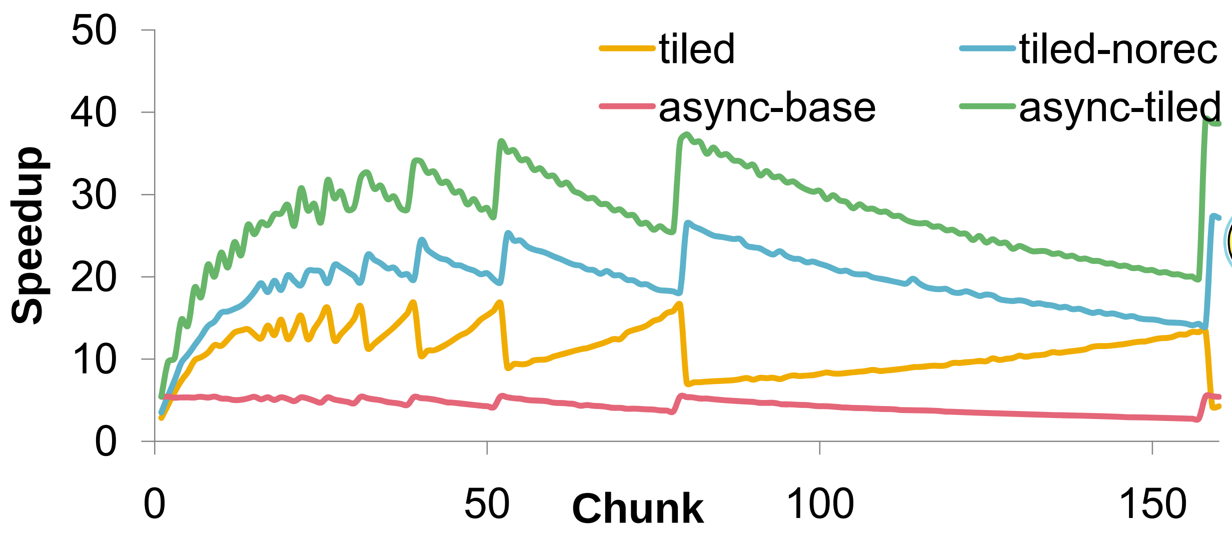
Configurations

- Peak bandwidth of our platforms

Machine	Model	L1	L2	L3	BW (GB/s)	SBW (GB/s)
A	AMD8350 4x4 cores	64KB private	512KB private	4x2MB shared	21.6	18.9
B	Q6600 1x4 cores	32KB private	2x4MB shared	N/A	8.5	4.79
C	E5530 2x4 cores	256KB private	1MB private	2x8MBs shared	51	31.5

Results - Jacobi

- Machine A



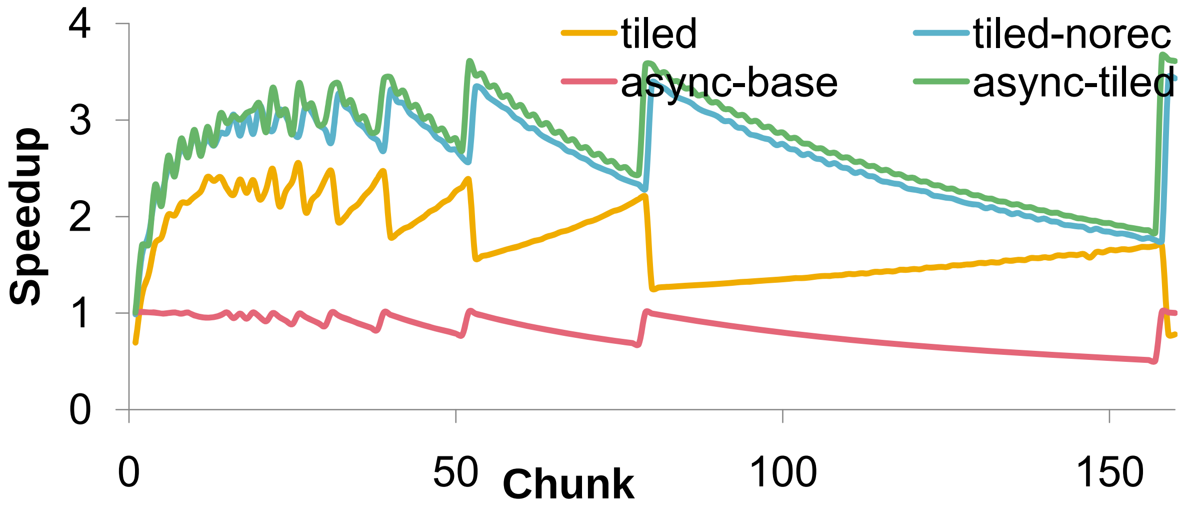
Performance varies with chunk size!

async-tiled version is the best!

Machine	kernel	parallel	tiled	tiled-norec	async-base	async-tiled
A 16 cores	Jacobi	5.95	16.76	27.24	5.47	39.11

Results - Jacobi

- Machine B

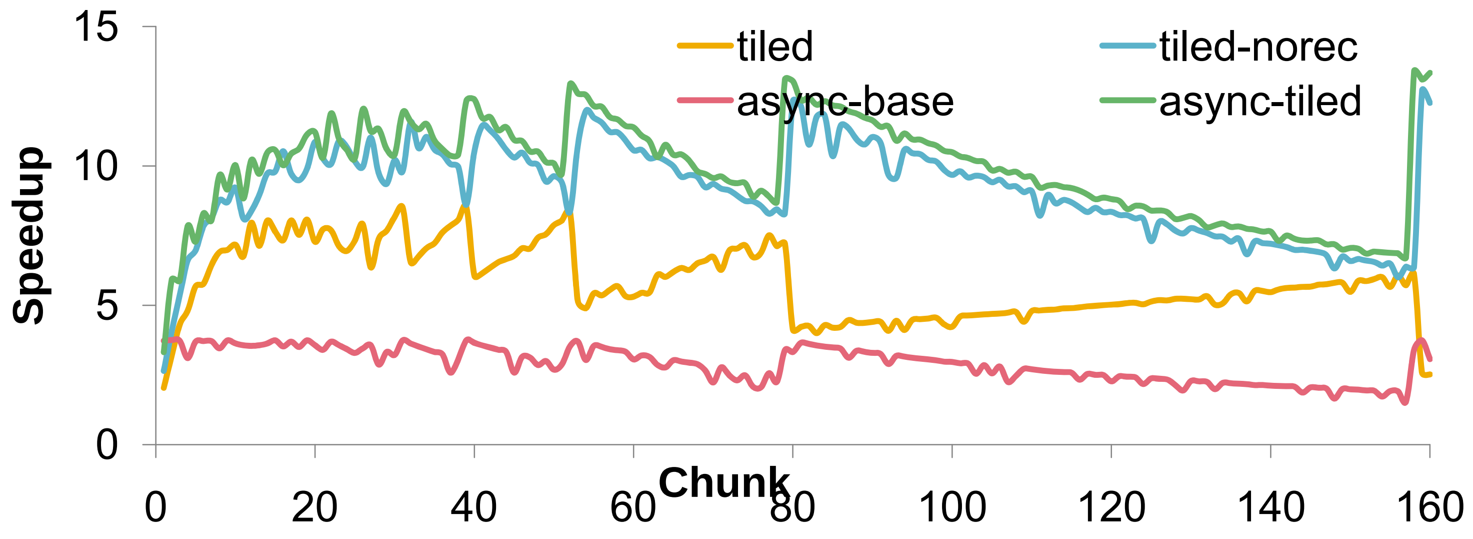


Poor performance without tiling (*async-base* and parallel)!

Machine	kernel	parallel	tiled	tiled-norec	async-base	async-tiled
B 4 cores	Jacobi	1.01	2.55	3.44	1.01	3.67

Results - Jacobi

- Machine C



Machine	kernel	parallel	tiled	tiled-norec	async-base	async-tiled
C 8 cores	Jacobi	3.73	8.53	12.69	3.76	13.39

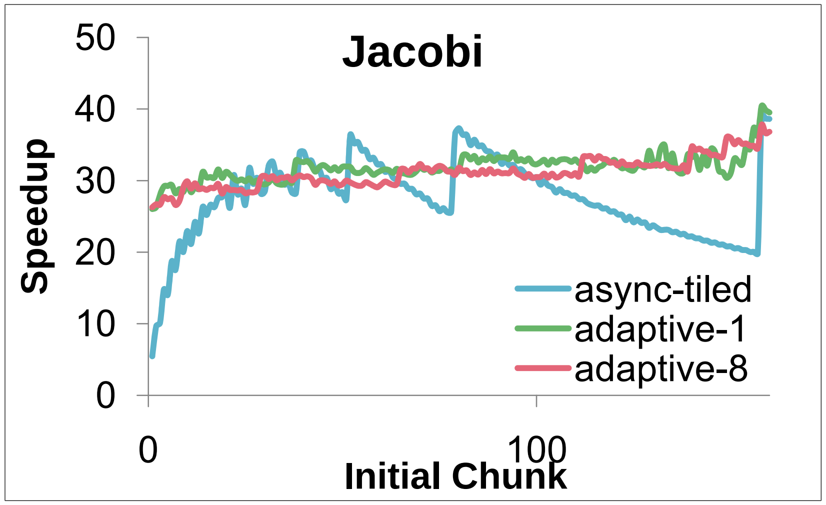
Results

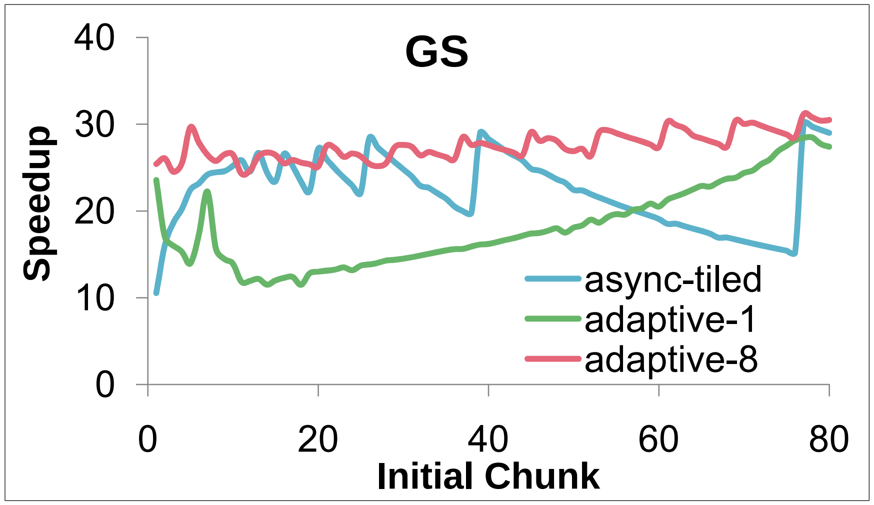
Machine	kernel	parallel	tiled	tiled-norec	async-base	async-tiled
A	GS	5.49	12.76	22.02	26.19	30.09
B	GS	0.68	5.69	9.25	4.90	14.72
C	GS	3.54	8.20	11.86	11.00	19.56
A	SOR	4.50	11.99	21.25	29.08	31.42
B	SOR	0.65	5.24	8.54	7.34	14.87
C	SOR	3.84	7.53	11.51	11.68	19.10

- Asynchronous tiled version performs better than synchronous tiled version (even without recovery cost)
- Asynchronous baseline suffers more on machine B due to less memory bandwidth available

Adaptive Chunk Size

- *adaptive-1*: lower bound of chunk size is *1*
- *adaptive-8:* lower bound of chunk size is *8*





Conclusions

- Showed how to benefit from the asynchronous model for relaxing data and control dependences
  - improve parallelism and data locality (via loop tiling) at the same time
- An adaptive method to determine the chunk size
  - because the iteration count is usually unknown in practice
- Good performance enhancement when tested on three well-known numerical kernels on three different multicore systems.

Q&A

- Thank you!