UPPSALA
UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

Efficient Implementation of a High-dimensional PDE-solver on Multicore Processors

Magnus Gustafsson
Sverker Holmgren

Uppsala University
Division of Scientific Computing

November 26, 2009

UPPSALA UNIVERSITET

Introduction

Spatial discretization

Temporal discretization

Minimizing communication

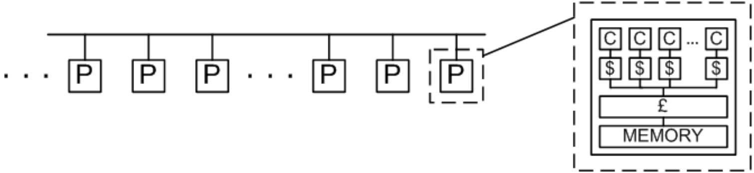
Conclusion

Framework for high-dimensional PDEs

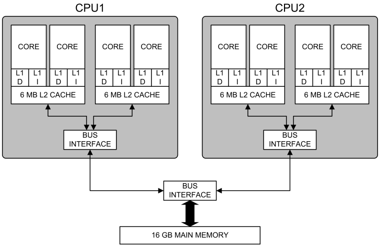
- Trade-off: Generality $\longleftrightarrow$ (Parallel) Efficiency
- Want a little bit of each:
  - Isolate independent components
  - Object-oriented philosophy
  - Choose performance-critical components at compile time
- Current (pilot) framework:
  - Implemented in C
  - Designed for clusters of multicore nodes
    - Message passing (MPI) between distributed nodes
    - OpenMP for worksharing within each node

Clusters of multicore nodes

- Grand scale computing required for realistic problems



- Example node architecture, dual Intel Xeon E5430:



Application: Quantum dynamics

- The Time-Dependent Schrödinger Equation (TDSE):

$$i\hbar\frac{\partial}{\partial t}\psi(\mathbf{r},t) = \hat{H}\psi(\mathbf{r},t)$$

$$\hat{H} = \hat{T} + \hat{V} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r},t)$$

- Models wave-packets moving over potential surfaces
- Several potential surfaces + collision with laser pulses ...
- Goal: To model basic chemical reactions

Application: Quantum dynamics

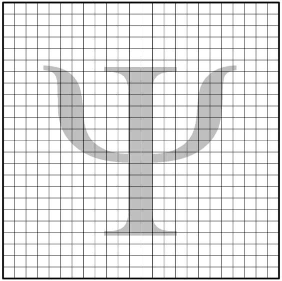
- Curse of dimensionality:

# particles	d = # spatial dimensions
2	1
3	3
4	6
5	9
$\vdots$	$\vdots$
n	$3n-6$

- Example (memory requirements of a 4-particle system):
  $d = 6$, $n_1 = \ldots = n_d = 100$, complex double precision
  $\Longrightarrow 100^6 * 16B = 10^{12} * 16B \approx \underline{16\,TB}$
  — *just to store the wavefunction!*

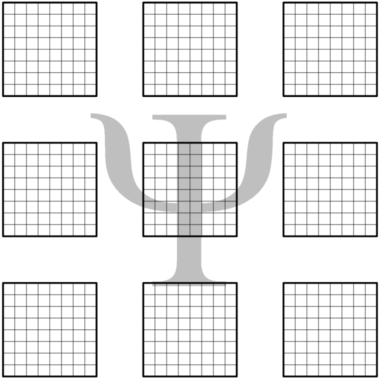
Spatial discretization

- Block-structured grid in *d* dimensions
  - Currently employing an equidistant, static grid
  - Choose block sizes w.r.t. cache sizes
  - Adaptive grid refinement/coarsening to be implemented

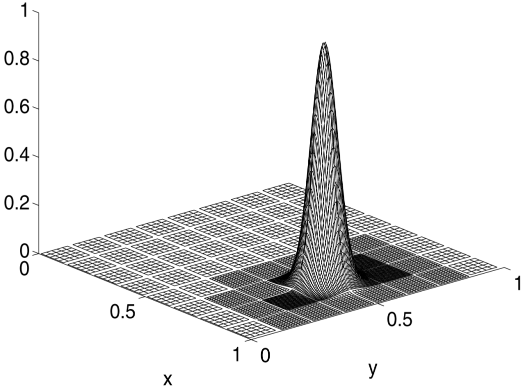


Spatial discretization

- Block-structured grid in d dimensions
  - Currently employing an equidistant, static grid
  - Choose block sizes w.r.t. cache sizes
  - Adaptive grid refinement/coarsening to be implemented



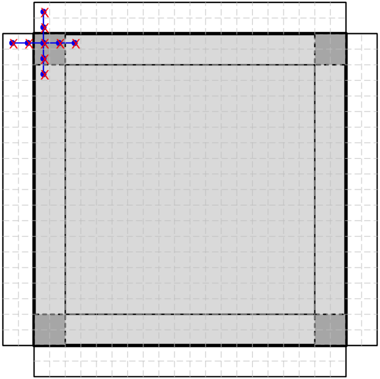
Outlook: Block-adaptive mesh refinement



Courtesy of J. Rantakokko, Uppsala University

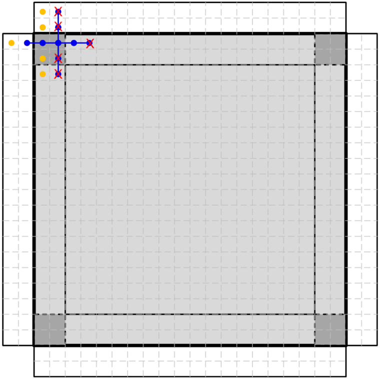
Spatial discretization

- High-order finite difference stencils



Spatial discretization

- High-order finite difference stencils

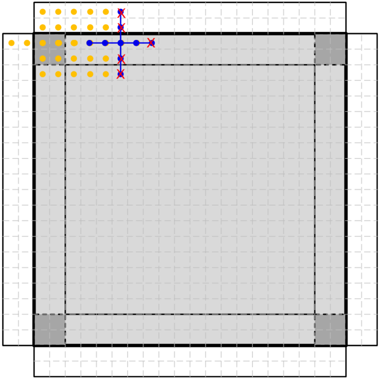


UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

Spatial discretization

- High-order finite difference stencils

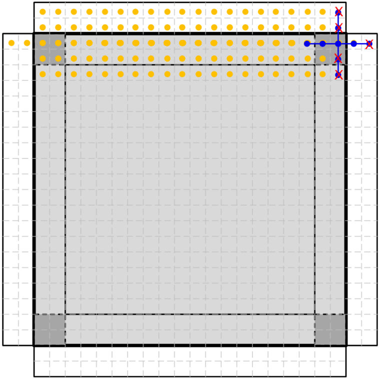


UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

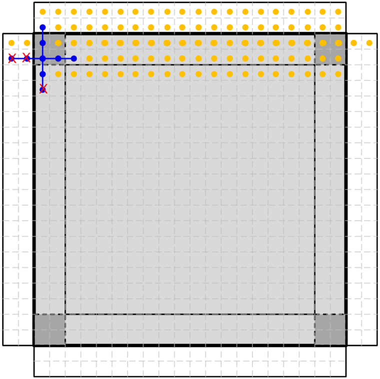
Spatial discretization

- High-order finite difference stencils



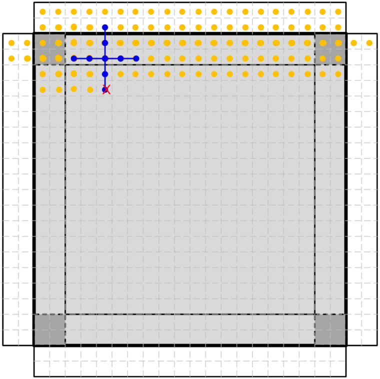
Spatial discretization

- High-order finite difference stencils



Spatial discretization

- High-order finite difference stencils



UPPSALA UNIVERSITET

Introduction

Spatial discretization

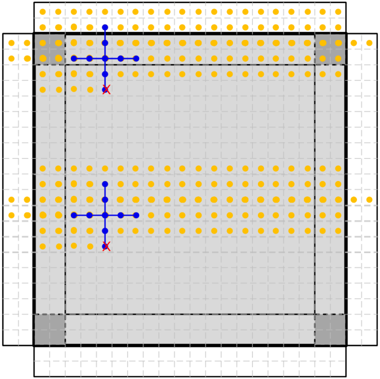
Temporal discretization

Minimizing communication

Conclusion

Spatial discretization

- High-order finite difference stencils

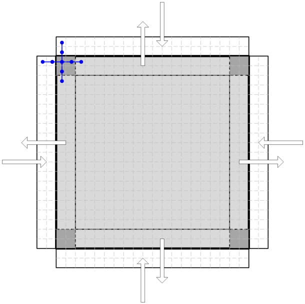


UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

Spatial discretization

- Data dependencies on block boundaries (ghost cells)
  - High-dimensional ghost cell blocks
  - Large memory overhead due to duplicated data
    - Communicate data one dimension at a time
    - Reuse allocated arrays for ghost data
  - Nearest-neighbor communication

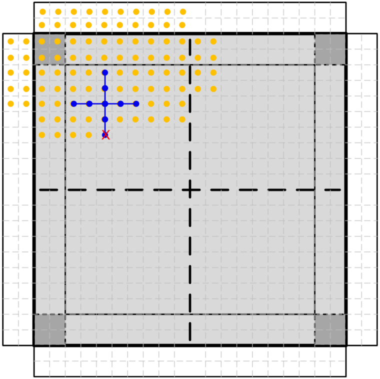


UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

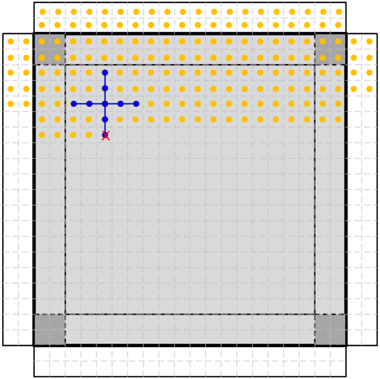
Blocking

- Separate into equally-sized blocks,do one block at a time
- Will destroy prefetch strides



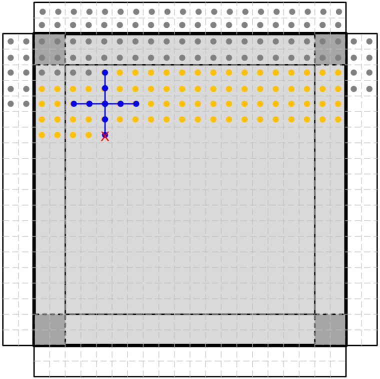
Tiling

- Partial blocking in the trailing dimension(s)
- Avoid breaking strides



Tiling

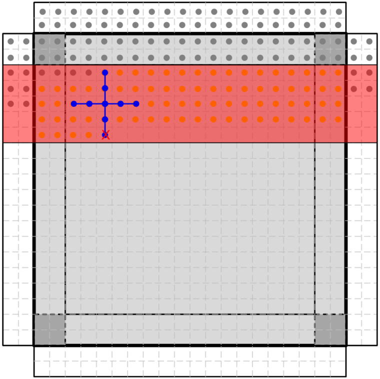
- Partial blocking in the trailing dimension(s)
- Avoid breaking strides



UPPSALA UNIVERSITET

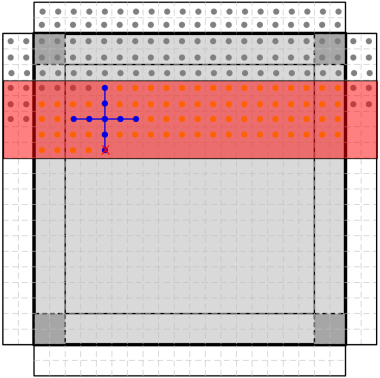
Tiling

- Partial blocking in the trailing dimension(s)
- Avoid breaking strides



Tiling

- Partial blocking in the trailing dimension(s)
- Avoid breaking strides

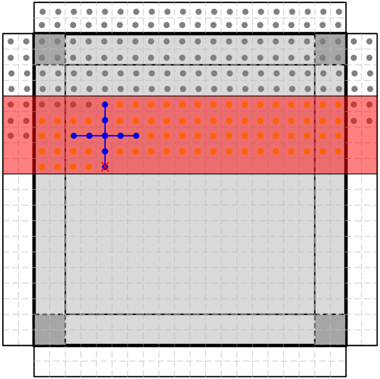


UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

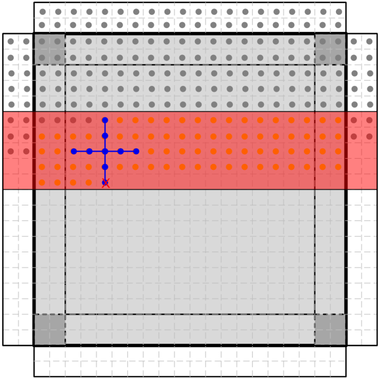
Tiling

- Partial blocking in the trailing dimension(s)
- Avoid breaking strides



Tiling

- Partial blocking in the trailing dimension(s)
- Avoid breaking strides



UPPSALA UNIVERSITET

Introduction

Spatial discretization

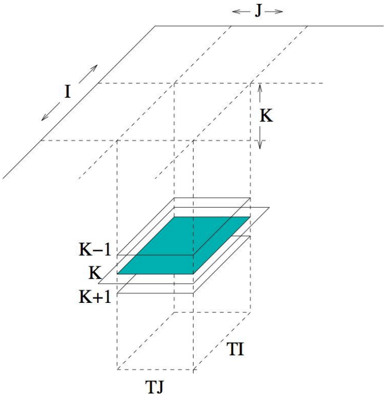
Temporal discretization

Minimizing communication

Conclusion

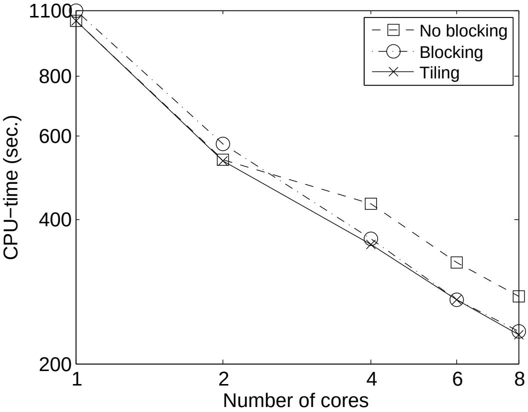
Tiling in 3D

- 2D tiles stacked on top of each other



Courtesy of Berkeley Benchmarking and Optimization group (BeBOP); `bebop.cs.berkeley.edu`

Impact of tiling



UPPSALA UNIVERSITET

Introduction

Spatial discretization

Temporal discretization

Minimizing communication

Conclusion

Temporal discretization

- The symmetric Lanczos algorithm
  - Approximates a few of the most extremal eigenvalues
  - Use this to compute e^{-iH} at low computational cost
- Difficult to achieve massive scalability, since in each iter.
  - Multiplication w. Hamiltonian matrix (nearest-neighbor)
  - Two inner products (all-to-all)

Algorithm 1 THE LANCZOS ALGORITHM

$\boldsymbol{v_0} = \mathbf{0}$
$\beta_0 = 0$
$\boldsymbol{v_1} = \boldsymbol{\Psi_k} \;/\; \|\boldsymbol{\Psi_k}\|_2$
for $j = 1, 2, \ldots, m$ **do**
  $\boldsymbol{r} = \boldsymbol{H v_j} - \beta_{j-1}\boldsymbol{v_{j-1}}$
  $\alpha_j = (\boldsymbol{r}, \boldsymbol{v_j})$
  $\boldsymbol{r} = \boldsymbol{r} - \alpha_j \boldsymbol{v_j}$
  if j < m **then**
    $\beta_j = \|\boldsymbol{r}\|_2$
    $\boldsymbol{v_{j+1}} = \boldsymbol{r} \;/\; \beta_j$
  end if
end for

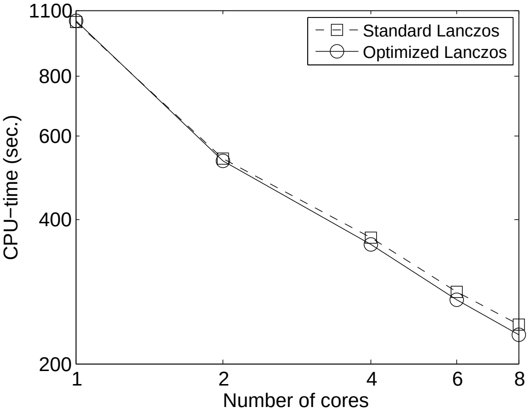
A modified Lanczos scheme

- According to Kim and Chronopoulos (1991):
  - Restructure Lanczos' algorithm and bring the two inner products together
  - Eliminates one synchronization point

Algorithm 2 THE MODIFIED LANCZOS ALGORITHM

$\boldsymbol{q_0} = \mathbf{0}$
$\boldsymbol{r_0} = \boldsymbol{\Psi_k} \;/\; \|\boldsymbol{\Psi_k}\|_2$
for $j = 0, 1, 2, \dots, m$ **do**
  $\boldsymbol{t} = \boldsymbol{Ar_j}$
  $\boldsymbol{u} = (\boldsymbol{t}, \boldsymbol{r_j})$
  $\boldsymbol{v} = (\boldsymbol{r_j}, \boldsymbol{r_j})$
  $\beta_j = \sqrt{\boldsymbol{v}}$
  $\alpha_{j+1} = \boldsymbol{u}/\boldsymbol{v}$
  $\boldsymbol{q_{j+1}} = \boldsymbol{r_j}/\beta_j$
  $\boldsymbol{r_{j+1}} = \boldsymbol{t}/\beta_j - \beta_j \boldsymbol{q_j} - \alpha_{j+1}\boldsymbol{q_{j+1}}$
end for

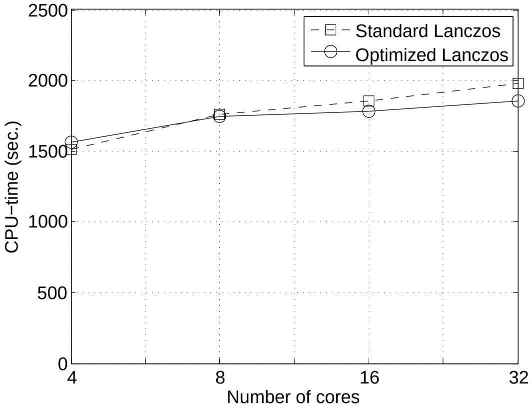
Performance of modified Lanczos'



UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

Performance of modified Lanczos'



Outlook: s-step Lanczos

- Krylov subspace methods compute an orthonormal basis where the vectors are computed one-by-one

$$span\{y, Ay, Ay^2, \ldots, A^{m-1}y\}$$

- What if we could compute several vectors at once?
  - cf. Demmel *et al.* (2008)
- Kim and Chronopoulos (1991) proved that the Lanczos algorithm can be reformulated in this way
  - Reduces the number of synch. points by a factor of s
  - Not implemented in parallel, but we have a working MATLAB version

Conclusion

- Node-local performance is key to overall performance; so that is where we need to optimize first
- Communication is expensive in modern parallel systems; we aim at minimizing it
- Massive scalability is hard to achieve; might have to rewrite old algorithms
- Future work:
  - Implement s-step Lanczos in parallel
  - Spatial adaptivity
  - Analyze the impact of thread placement and scheduling

UPPSALA UNIVERSITET

Introduction
Spatial discretization
Temporal discretization
Minimizing communication
Conclusion

References

S. K. Kim and A. T. Chronopoulos.
A Class of Lanczos-like Algorithms Implemented on Parallel Computers
Parallel Computing, 17 (1991).

J. Demmel, M. Hoemmen, M. Mohiyuddin and K. Yelick
Avoiding Communication in Sparse Matrix Computations
Proceedings of IEEE International Parallel and Distributed Processing Symposium, April, 2008