UPPSALA
UNIVERSITET

Sparse Matrix-Vector Multiplication and Matrix Formats

Dimitar Lukarski

Division of Scientific Computing
Department of Information Technology
Uppsala Programming for Multicore Architectures Research Center
(UPMARC)
Uppsala University

Parallel Scientific Computing, April 11, 2013

Outline

UPPSALA UNIVERSITET

Parallel Computing

- Parallel hardware is everywhere!
- Phones, Tablets, PCs, GPUs, Xbox, PS, ... TV!
- Good parallel programming is not easy
- Parallel programs could be very fast!
- This is a growing market (and need)
- CPU+Accelerator(GPU/MIC) delivers high perf

Why GPU Computing?

Comparison to UPPMAX

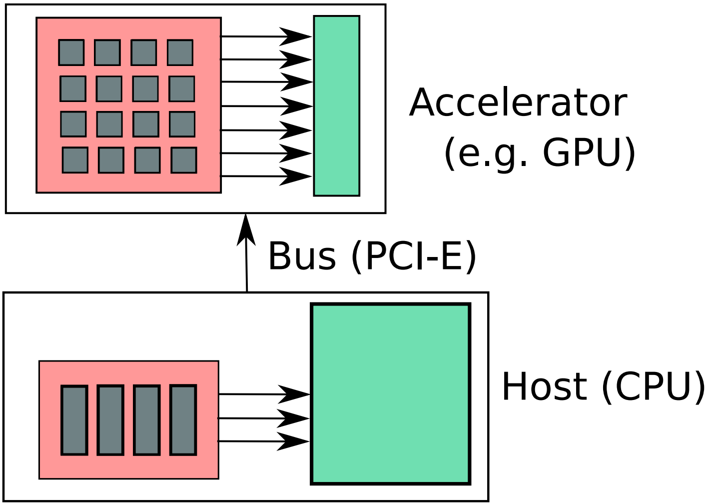
- Tintin: 160 $\times$ dual Opteron 6220
- Each Opteron 6220 = 192 GFlop/s
- In total (theoretical) = 61.4 TFlop/s
- 7.6TFlop/s if you have serial code in the nodes

Equivalent GPU-based configuration

- K20 device = 1.17 TFlop/s
- 53 cards = 62 TFlop/s
- 14 nodes with 4 GPUs (56 cards)

UPPSALA UNIVERSITET

Accelerators in the Computer



Accelerators in the Computer

Bandwidth

- Accelerator memory - very fast
- Host memory - fast
- PCI-E bus - slow
- Network - slow
- Hard disk - very slow

Capacity

- Hard disk - very large
- Host memory - large
- Accelerator memory - small (2-8GB)

Compute capabilities

- Host CPU - few fat cores (they do everything!)
- Accelerator chip - many, small, specialized (Flop/s)

UPPSALA UNIVERSITET

Introduction and Motivation

Sparse Matrices

Matrix Formats

Sparse Matrix-Vector Multiplication

Parallel Sparse Matrix-Vector Multiplication

Performance

Take away message

Scalability on Multi/Many-core

UPPSALA
UNIVERSITET

Sparse Matrices

- Continuous problem (PDE)
- Discretize schemes - FD, FE, FV
- **Sparse (non-)linear problem**
- Linear solver
- Solution

Sparse matrix is a matrix (real, complex) where most of the elements are zeros: $A \in R^{N,N}$ then the number non-zero elements (NNZ) is O(N).

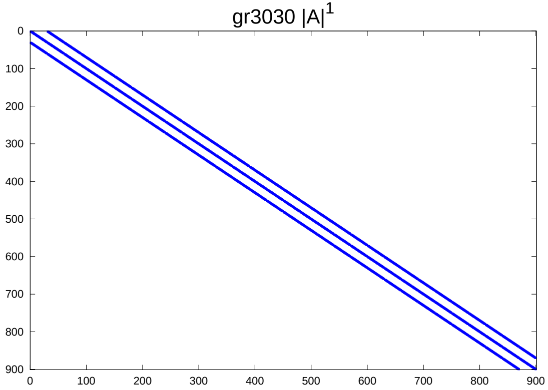
Sparsity Patterns

- Mesh type
  - Elements
  - Structured / un-structured
- Problem dimension (2D, 3D)
- Discretization method
- Order of the scheme
- Ordering method

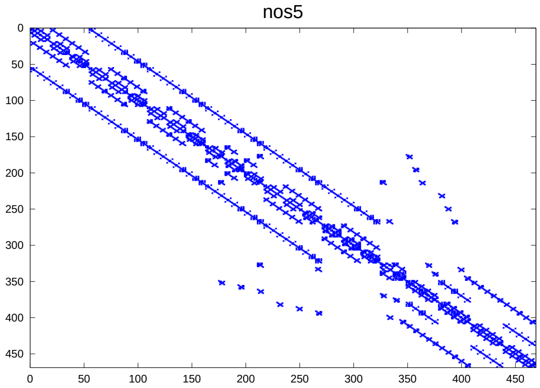
Example: FD, Laplace, $A \in R^{N,N}$, lexicographical order

- 1D $N = n$, 3 diagonals (-1,2,-1)
- 2D $N = n \times n$, 5 diagonals (-1,-1,4,-1,-1)
- 3D $N = n \times n \times n$, 7 diagonals (-1,-1,-1,6,-1,-1,-1)

Example: Finite-difference Laplacian



Example: Structural Mechanics



Test Matrix

$A \in R^{5,5}$, NNZ=11, no pattern

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 1 & 2 & & 11 & \\
1 & & 3 & 4 & & \\
2 & & 5 & 6 & 7 & \\
3 & & & & 8 & \\
4 & & & & 9 & 10
\end{array}
$$

Matrix Formats - COO

Coordinate format:

- Row index (int) (NNZ)
- Column index (int) (NNZ)
- Values (data type) (NNZ)

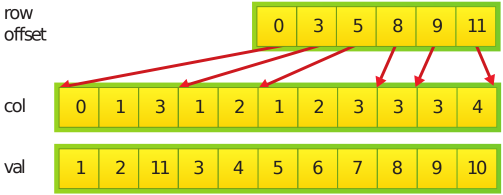
row	0	0	0	1	1	2	2	2	3	4	4
col	0	1	3	1	2	1	2	3	3	3	4
val	1	2	11	3	4	5	6	7	8	9	10

UPPSALA UNIVERSITET

Matrix Formats - CSR

Compressed Sparse Row format:

- Row offsets (int) (N+1)
- Column index (int) (NNZ)
- Values (data type) (NNZ)



Analogous CSC (Compressed Sparse Column)

Matrix Formats - ELL

ELL format:

- Column index (int) (N*M)
- Values (data type) (N*M)
- where M is the max number of el per row

col

0	1	3
1	2	*
1	2	3
3	*	*
3	4	*

val

1	2	11
3	4	*
5	6	7
8	*	*
9	10	*

Matrix Formats - DIA

Diagonal format:

- Diagonal offsets (Ndiag)
- Values (data type) (N*Ndiag)

dig

-1	0	1	3

val

*	1	2	11
0	3	4	0
5	6	7	*
0	8	0	*
9	10	*	*

Matrix Formats - Memory Footprint

Format	Structure	Values
Dense	–	$N \times N$
COO	$2 \times NNZ$	NNZ
CSR	$N + 1 + NNZ$	NNZ
ELL	$M \times N$	$M \times N$
DIA	D	$D \times N_D$

Matrix Formats - HYB

Almost Perfect Pattern

- Major part of the elements have pattern
- Only few elements do not belong to the pattern

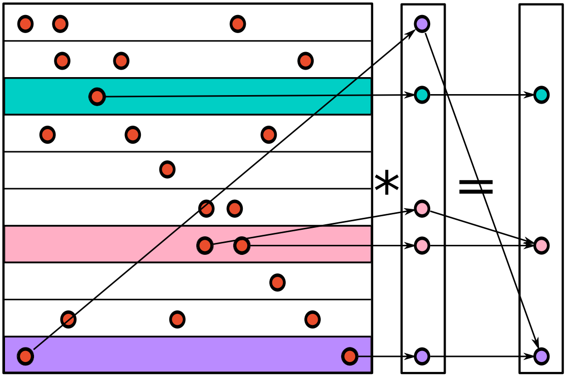
$A := B + C$

- Use different format for B and C
- Example: ELL/DIA + COO

UPPSALA UNIVERSITET

UPPSALA UNIVERSITET

Sparse Matrix-Vector Multiplication y=Ax



SpMV - COO

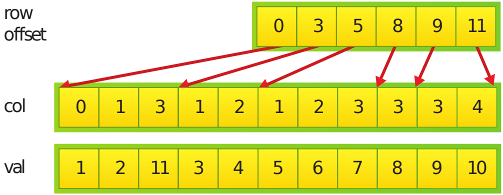
```
for (int i=0; i<n; ++i)
 y[i] = 0.0;

for (int i=0; i<nnz; ++i)
 y[row[i]] += val[i]*x[col[i]];
```

row	0	0	0	1	1	2	2	2	3	4	4
col	0	1	3	1	2	1	2	3	3	3	4
val	1	2	11	3	4	5	6	7	8	9	10

SpMV - CSR

```
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=row_off[i]; j<row_off[i+1]; ++j)
  y[i] += val[j]*x[col[j]];
}
```



Transpose use CSC

SpMV - ELL

```
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=0; j<max_row; ++j) {
  jj = j + max_row*i;
  c = col[jj]
  if ((c >= 0) && (c < n))
   y[i] += val[jj] * x[c];
  }
}
```

col

0	1	3
1	2	*
1	2	3
3	*	*
3	4	*

val

1	2	11
3	4	*
5	6	7
8	*	*
9	10	*

SpMV - DIA

dig

-1	0	1	3

val

*	1	2	11
0	3	4	0
5	6	7	*
0	8	0	*
9	10	*	*

SpMV - DIA

```
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=0; j<n_diag; ++j) {
 int start, v_offset, end = n_row();
 if (diag[j] < 0) {
  start -= diag[j];
  v_offset = -start;
 } else {
  end -= diag[j];
  v_offset = diag[j];
 }
 ind = j*n_row + i;
 if ( (i >= start) && (i < end)) {
   y[i] += val[ind] * x[i+v_offset];
  }
}
```

UPPSALA UNIVERSITET

Matrix Foramts - HYB

$y := Ax$, where $A := B + C$
is
$y := Bx + Cx$

UPPSALA UNIVERSITET

UPPSALA UNIVERSITET

Shared/Distributed Memory Systems

Shared memory systems

- All cores have the same memory access
- Fine-grained level of parallelism
- Memory access pattern

Distributed memory systems

- All nodes are connected via network
- Coarse-grained level of parallelism
- Communication pattern

Outline
Intro and Motivation
Sparse Matrices
Matrix Formats
SpMV
Parallel SpMV
Performance
Conclusion
Extra Notes

SpMV on Shared Memory Systems

CPU (x86)

- OpenMP

SpMV - COO

```
#pragma omp parallel for
for (int i=0; i<n; ++i)
 y[i] = 0.0;

???
for (int i=0; i<nnz; ++i)
 y[row[i]] += val[i]*x[col[i]];
```

- Segmented/Prefix scan
- Sort

UPPSALA UNIVERSITET

SpMV - CSR

```
#pragma omp parallel for
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=row_off[i]; j<row_off[i+1]; ++j)
  y[i] += val[j]*x[col[j]];
}
```

- The elements per row could be sorted or not
- Transpose in CSC or with atomics

UPPSALA UNIVERSITET

SpMV - ELL

```
#pragma omp parallel for
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=0; j<max_row; ++j) {
  jj = j + max_row*i;
  c = col[jj]
  if ((c >= 0) && (c < n))
   y[i] += val[jj] * x[c];
  }
}
```

- the elements per row could be sorted or not

SpMV - DIA

```
#pragma omp parallel for
for (int i=0; i<n; ++i) {
 y[i] = 0.0;
 for (int j=0; j<n_diag; ++j) {
 int start, v_offset, end = n_row();
 if (diag[j] < 0) {
  start -= diag[j];
  v_offset = -start;
 } else {
  end -= diag[j];
  v_offset = diag[j];
 }
 ind = j*n_row + i;
 if ( (i >= start) && (i < end)) {
   y[i] += val[ind] * x[i+v_offset];
  }
}
```

SpMV on Shared Memory Systems

GPU (or any many-core device)

- CUDA

SpMV - COO

- Pre-sorted
- Segmented reduction

SpMV - CSR

```
template <typename ValueType, typename IndexType>
__global__ void spmv(const IndexType nrow,
 const IndexType *col, const ValueType *val,
 const ValueType *in, ValueType *out) {

 IndexType ai = blockIdx.x*blockDim.x
                 +threadIdx.x;
 IndexType aj;

 if (ai <nrow) {

  out[ai] = ValueType(0.0);

  for (aj=row_off[ai]; aj<row_off[ai+1]; ++aj) {
    out[ai] += val[aj]*in[col[aj]];
  }
 }
```

SpMV - CSR

- Straightforward implementation
- Each thread - one row
- Load imbalance
- Large number of el per row - vector impl

SpMV - ELL

Index mapping

- CPU: el + max_row*row
- GPU: el*nrow + row

Performance

- Each thread - one row
- Constant (*almost*) load for each thread

SpMV - DIA

Index mapping

- CPU: el + ndiag*row
- GPU: el*nrow + row

Performance

- Each thread - one row
- Constant (*almost*) load for each thread

UPPSALA UNIVERSITET

SpMV on Distributed Memory Systems

Clusters

- The mesh is distributed
- Each node contains part of the mesh
- Each node know the distribution

Domain Decomposition

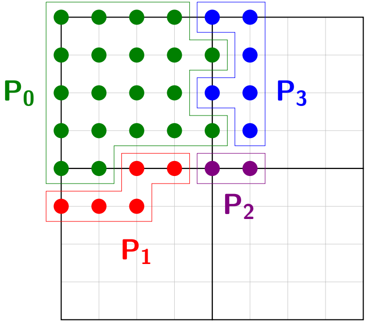
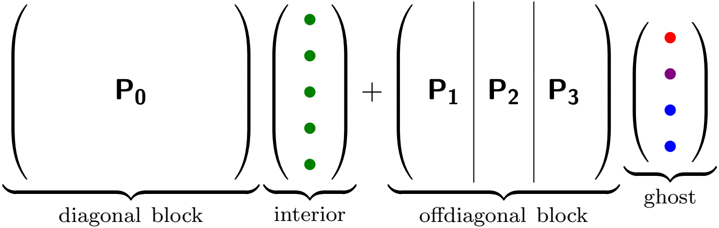


Figure: Domain partitioning: DOF of process $\mathbf{P_0}$ are marked in green (interior DOF in diagonal block); the remaining DOF represent inter-process couplings for process $\mathbf{P_0}$ (ghost DOF in off-diagonal block)

UPPSALA
UNIVERSITET

SpMV Distribution and Communication



Sparse Matrix-Vector Multiplication:

- start asynchronous communication
- exchange ghost values
- $y_{\mathrm{int}} = A_{\mathrm{diag}}\, x_{\mathrm{int}}$
- synchronize communication
- $y_{\mathrm{int}} = y_{\mathrm{int}} + A_{\mathrm{offdiag}}\, x_{\mathrm{ghost}}$

UPPSALA UNIVERSITET

SpMV on Distributed Memory Systems

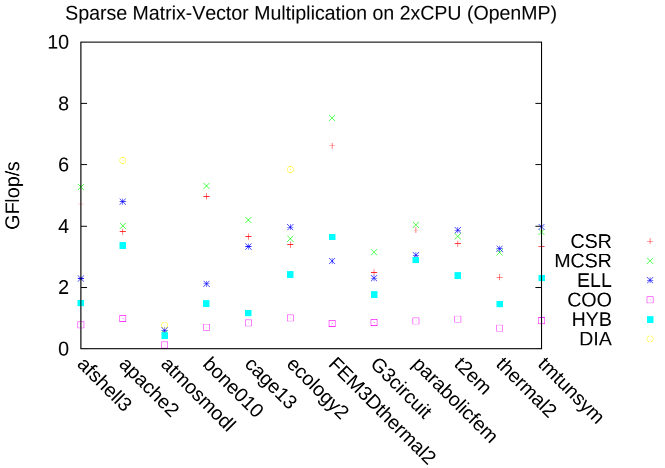
Clusters (Accelerators/GPUs)

- Each accelerator is connected via PCI-E
- Network speed $\approx$ PCI-E speed
- Use async transfers over the PCI-E
- Use **blocking** technique for the ghost layers

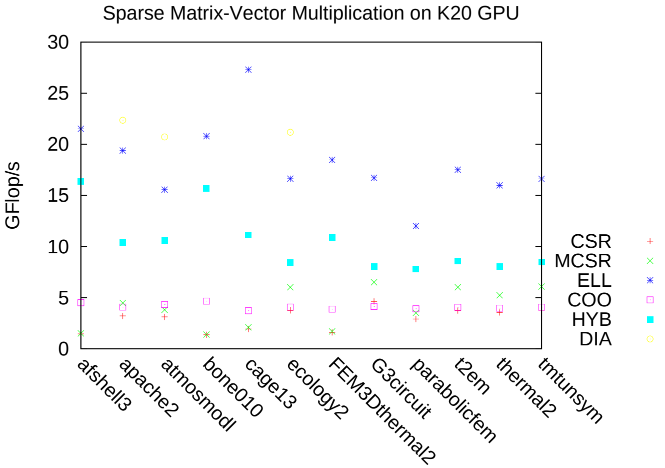
SpMV Multi-core/GPU

- 2x Xeon E5-2680 (8cores + HT)
- 1 x GPU K20
- OpenMP 16 threads
- CUDA

SpMV - 16 threads CPU



SpMV - GPU


Sparse Matrix-Vector Multiplication on K20 GPU
GFlop/s
0
5
10
15
20
25
30
afshell3
apache2
atmosmodl
bone010
cage13
ecology2
FEM3Dthermal2
G3circuit
parabolicfem
t2em
thermal2
tmtunsym
CSR
MCSR
ELL
COO
HYB
DIA

UPPSALA UNIVERSITET

Distributed CG solver

3D-Neumann-Laplace problem:

$$-\Delta \mathbf{u} = \mathbf{f}, \quad \text{in } \Omega = [0,1]^3,$$
$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = 0, \quad \text{on } \partial\Omega,$$

where $\mathbf{f}$ given rhs such that $\int_\Omega \mathbf{f}\, d\mathbf{x} = \mathbf{0}$.

Weak formulation:

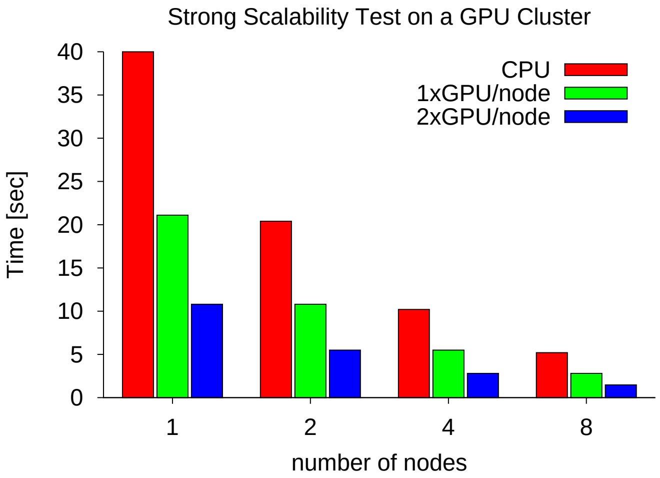
$$(\nabla \mathbf{u}, \nabla \varphi) = (\mathbf{f}, \varphi), \quad \text{for all } \varphi \in \mathbf{H}^1$$

GPU cluster

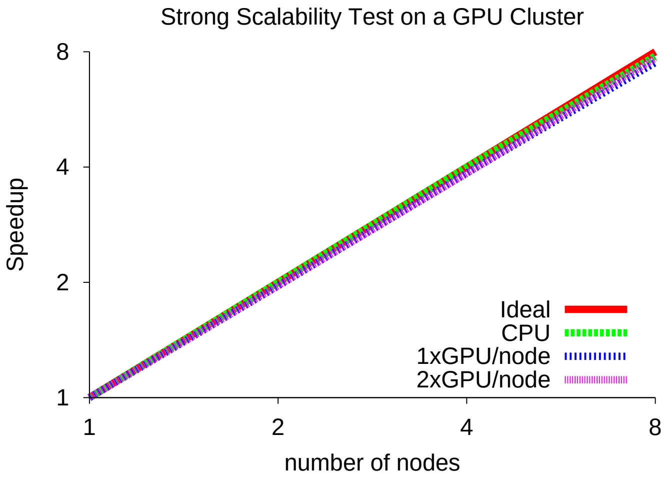
- 8 nodes with two Xeon 5500 (Nehalem)
- Interconnect - 20 Gbit/s Infiniband fabric
- Small DDR network switch
- Two NVIDIA Tesla M1060 GPU per node

UPPSALA UNIVERSITET

CG solver



CG solver


Strong Scalability Test on a GPU Cluster
Speedup
8
4
2
1
1
2
4
8
number of nodes
Ideal
CPU
1xGPU/node
2xGPU/node

UPPSALA UNIVERSITET

Introduction and Motivation

Sparse Matrices

Matrix Formats

Sparse Matrix-Vector Multiplication

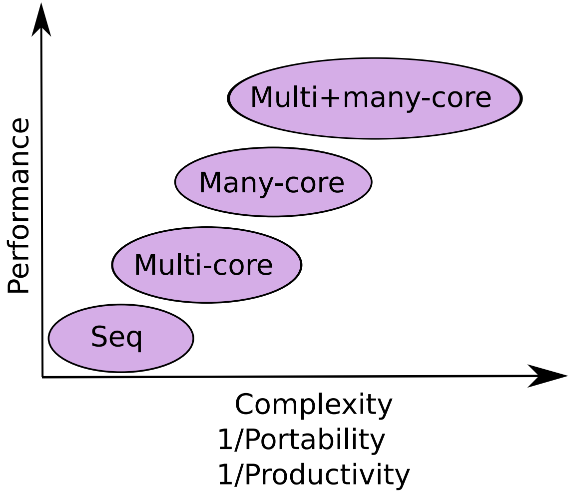
Parallel Sparse Matrix-Vector Multiplication

Performance

Take away message

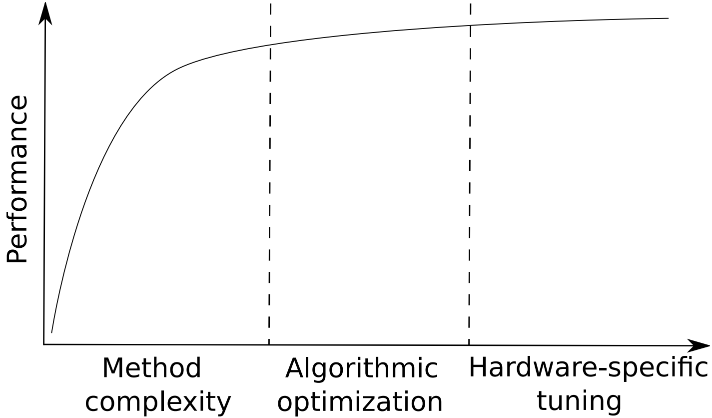
Scalability on Multi/Many-core

Good Parallel Programming is Hard



UPPSALA UNIVERSITET

Methods are the MOST Imporant



Scalability on Multi/Many-core

Distributed systems

- More nodes = more GFlop/s
- More nodes = more GByte/s

Multi-core systems

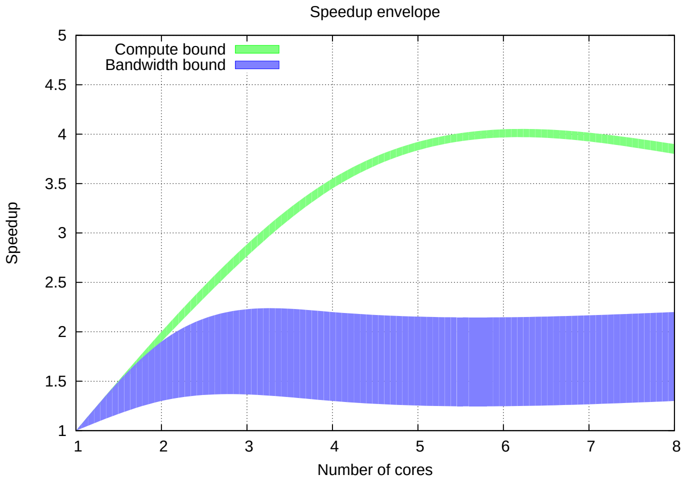
- More cores = more GFlop/s
- More cores = little extra GByte/s

GPU systems

- Thousand threads = peak GFlop/s
- Thousand threads = peak GByte/s

UPPSALA UNIVERSITET

Typical Speed-up Numbers



- i7 SandyBridge (4cores+4HT)

Computational Complexity of Algorithms