SPAI preconditioners for HPC applications

G. Fourestey (CSCS/ETHZ), W. Sawyer (CSCS/ETHZ), R. Popescu (EPFL), C. Vanini (USI)
What is HPC

What exactly is High-Performance Computing?

- “... it depends on who you talk to” Richard Dracott (Intel)
- “At IBM we called it scientific computing” Ulla Thiel (Cray)

Scientific computing where computer performance matters, in other words, scientific applications that need a lot of computer performance.
Exascale computing is: 1 billion billion sustained FP operations per second (measured using HPL).

Will exascale science exist?
Path to Exascale

Application performance seems to keep up with supercomputing systems performance (!)

- ~100 Kilowatts
- ~5 Megawatts
- 20-30 MW
- ~1 Exaflop/s

1.35 Petaflop/s
Cray XT5
150,000 processors

1.02 Teraflop/s
Cray T3E
1,500 processors

1 Gigaflop/s
Cray YMP
8 processors

<table>
<thead>
<tr>
<th>1988</th>
<th>1998</th>
<th>2008</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>First sustained GFlop/s</td>
<td>First sustained TFlop/s</td>
<td>First sustained PFlop/s</td>
<td>Another 1,000x in sustained performance increase</td>
</tr>
</tbody>
</table>

Monday, November 21, 2011
CPU Trends

Until 2005, code performance scale wrt the CPU frequency (hardware).

First physical roadblock: CPU consumption $\sim \text{freq}^{\wedge}3$!
1st Level of Parallelism

MPI  MPI  MPI  MPI
CPU Trends

After 2005, code performance scales wrt the number of core.
Petascale computing was driven by multi-core technology.

- MPI is at the center of Petascale technology
- Some software (scalability) and hardware (IO) had to be sorted out, but petascale science is working.

Is this good enough to reach Exascale computing?

Unfortunately, we will hit a new road blocks.
## Path to Exascale

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Peak Flops/s</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>System Memory</td>
<td>0.3 PB</td>
<td>1 PB</td>
<td>5 PB</td>
<td>10 PB</td>
</tr>
<tr>
<td>Node Performance</td>
<td>125 GF</td>
<td>200 GF</td>
<td>400 GF</td>
<td>1-10 TF</td>
</tr>
<tr>
<td>Node Memory BW</td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>100 GB/s</td>
<td>200-400 GB/s</td>
</tr>
<tr>
<td>Node Concurrency</td>
<td>12</td>
<td>32</td>
<td>0(100)</td>
<td>0(1000)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>10 GB/s</td>
<td>25 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>System Size (Nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
<td>0(Million)</td>
</tr>
<tr>
<td>Total Concurrency</td>
<td>225,000</td>
<td>3 Million</td>
<td>50 Million</td>
<td>0(Billion)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
<td>300 PB</td>
</tr>
<tr>
<td>I/O</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>10 TB/s</td>
<td>20 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>Days</td>
<td>Days</td>
<td>Days</td>
<td>0(1Day)</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~10 MW</td>
<td>~10 MW</td>
<td>~20 MW</td>
</tr>
</tbody>
</table>
Power Consumption

Question: what is faster?

1.23456789101112 +
3.1415 *
1.12111098765432 +
Power Consumption

Question: what is faster?

\[
\begin{align*}
1.23456789101112 + \\
&3.1415 * \\
1.12111098765432 +
\end{align*}
\]

- \(\sim 1\) ns

\(\sim 100\) ns

Question: what is the more power hungry?
Question: what is faster?

\[ 1.23456789101112 + 3.1415 \times 1.12111098765432 + \]

- \(~1\) ns
- \(~100\) ns

Question: what is the more power hungry?

- \(X_1\)
- \(X_3\)

Conclusion: performance is not a question of flops but data movements, which are very expensive (time and power consumption)!
On a CPU, ~70% of the power consumption is memory management.

A solution? Not sure yet but high performance/power consumption ratio SIMD accelerators (GPU, MIC...) seem to be the way to go...
2nd Level of Parallelism

- MPI
- PCIe
- OpenMP

Core #0
Core #1
Core #2
Core #3

Shared L3 Cache
3rd Level of Parallelism

- MPI
- PCIe
- OpenMP

2013?
Exascale Brutal Facts

• Exascale scientific codes will have to:
  • to deal with millions (poss. billions) of threads,
  • use less MPI taks, possibly one per node,
  • less and slower memory per thread,
  • Small improvements in inter-processor and inter-thread communications,
  • Stagnant I/O subsystems,
  • take advantage of accelerators such as GPUs/manycore to maximize flops/watt,
  • redesign the code to use Hardware/Software co-degisn, such as heterogeneous models,
  • extract 3 levels of parallelism: MPI, OpenMP and vectorization (SIMD units, GPUs, ...)

Monday, November 21, 2011
Using MSPAI: Goals

The primary goal of MSPAI is to provide a High Performance preconditioner for the HP2C project LifeV, which is:

- a C++ open source HPC Library for Finite Elements computation, in particular for biomedical applications (FSI, multi-scale, multi-physics ...)
- developed at mainly EPFL (CH), MOX (IT), Emory Univ. (USA)

The serial core was developed at INRIA (FR), EPFL, MOX. The core parallel version was developed at EPFL (S. Deparis, G. Fourestey @CMCS, Pr. A. Quarteroni).

Lifev uses Trilinos for linear algebra solvers. See www.lifev.org.
Trilinos

“The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.”

- C++ based code,
- Matrix Classes, preconditioners, solvers ... 
- a lot different packages, most of them are independant or interchangeable, 
- complete MPI framework for communications.
Trilinos Preconditioners

Trilinos is using a very powerful class derivation system which allows the usage of third party codes (UMFPack, SuperLU ...)

Trilinos has several types of preconditioners:

- Block preconditioners: Meros
- ILU factorization: Ifpack + AztecOO
- Multigrid methods: ML
- Domain decomposition: Ifpack
Point relaxation preconditioners

- Jacobi
- Gauss-Seidel
- symmetric Gauss-Seidel

Block relaxation preconditioners

- Jacobi
- block Gauss-Seidel
- block symmetric Gauss-Seidel

Point incomplete factorizations.

- Exact factorizations:
  - all Amesos classes can be used to compute the LU factorization of a given matrix.
- Chebyshev polynomials.
Our goal is to create a new package, e.g. Inverse Approximation, that could be used in the Trilinos preconditioners framework.

- General preconditioner used in Ifpack for domain decomposition,
- Schur complement approximation,
- Smoother for ML,
- ...

Now, let’s link MSPAI and Trilinos.
MSPAI Implementation

The MSPAI code was standalone:

- it compiled an executable
- reading matrix and rhs in Matrix Market format
- the matrix columns were cut in contiguous chunks

First step: turn MSPAI code into a library

- the code compiled a library file as well as the executable
- Insert a namespace “mspai”
MSPAI Implementation

Second step: make the code accept another matrix format:

- Use Epetra_Map: each local map is saved in a file called `<matrix name>_<rank #>`
- Matrix values are read from the matrix market file then transformed into the local format/pattern.

Now, using EpetraExt::RowMatrixToMatrixMarket and Epetra_Map exporters, MSPAI can use Epetra_CrsMatrix matrices
MSPAI Implementation

Third step: modification in the data structure, each process has to know where each column is:

- each process is reading its own map file only!
- allreduce of this array (only once at the beginning)

all processes are aware of their next columns as well as what process has a specific column.
Ifpack_Preconditioner Class

Public Member Functions:

- virtual int SetParameters (Teuchos::ParameterList &List)=0 Sets all parameters for the preconditioner.
- virtual int Initialize ()=0 ; Computes all it is necessary to initialize the preconditioner.
- virtual int Compute ()=0 ; Computes all it is necessary to apply the preconditioner.
- virtual bool virtual int ApplyInverse (const Epetra_MultiVector &X, Epetra_MultiVector &Y) const =0 ; Applies the preconditioner to vector X, returns the result in Y.
Ifpack_Preconditioner Class

Public Member Functions:

- virtual int SetParameters: store the MSPAI parameters (eps, level, ...),
- virtual int Initialize: transforms the Epetra Matrix in the MSPAI format and performs the initial computations (Pattern, remote cols...),
- virtual int Compute: the actual computation calling the SPAI_Algorithm routine,
- virtual bool virtual int ApplyInverse: “applying” the preconditioner, i.e. doing the matrix vector multiply:

\[ M^*(Ax - y) \]
Ifpack_Preconditioner Class

Now it is possible to dynamically cast this class into Epetra_Operator and use it as:

- a general preconditioner for AztecOO (Gmres),
- a smoother for ML,
- ...

Monday, November 21, 2011
### Numerical Results

<table>
<thead>
<tr>
<th>MPI ranks</th>
<th>SPAI Comp. Time</th>
<th>SPAI GMRES Time</th>
<th>SPAI GMRES iter.</th>
<th>UMFPACK Comp. Time</th>
<th>UMFPACK GMRES Time</th>
<th>UMFPACK GMRES Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4.3</td>
<td>7.76</td>
<td>107</td>
<td>5.05</td>
<td>5.3</td>
<td>25</td>
</tr>
<tr>
<td>32</td>
<td>1.6</td>
<td>1.72</td>
<td>107</td>
<td>1.61</td>
<td>2.67</td>
<td>28</td>
</tr>
<tr>
<td>64</td>
<td>0.7</td>
<td>0.98</td>
<td>107</td>
<td>0.62</td>
<td>1.40</td>
<td>29</td>
</tr>
<tr>
<td>128</td>
<td>0.37</td>
<td>0.55</td>
<td>107</td>
<td>0.27</td>
<td>0.73</td>
<td>31</td>
</tr>
<tr>
<td>256</td>
<td>0.21</td>
<td>0.32</td>
<td>107</td>
<td>0.11</td>
<td>0.42</td>
<td>34</td>
</tr>
<tr>
<td>512</td>
<td>0.14</td>
<td>0.23</td>
<td>107</td>
<td>0.04</td>
<td>0.25</td>
<td>37</td>
</tr>
<tr>
<td>1024</td>
<td>0.05</td>
<td>0.18</td>
<td>107</td>
<td>0.02</td>
<td>0.16</td>
<td>41</td>
</tr>
<tr>
<td>2048</td>
<td>0.03</td>
<td>0.23</td>
<td>107</td>
<td>0.02</td>
<td>0.12</td>
<td>45</td>
</tr>
</tbody>
</table>

Comparison for AAS with MSPAI and UMFPack for the ADR problem, 216K dofs

---

Monday, November 21, 2011
## Numerical Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>69.13</td>
<td>11.27</td>
<td>283</td>
<td>5.28</td>
<td>7.94</td>
<td>11.27</td>
</tr>
<tr>
<td>32</td>
<td>46.66</td>
<td>7.74</td>
<td>272</td>
<td>2.91</td>
<td>6.46</td>
<td>7.74</td>
</tr>
<tr>
<td>64</td>
<td>37.15</td>
<td>5.65</td>
<td>262</td>
<td>2.16</td>
<td>6.18</td>
<td>5.65</td>
</tr>
<tr>
<td>128</td>
<td>32.15</td>
<td>4.81</td>
<td>260</td>
<td>1.47</td>
<td>5.38</td>
<td>4.81</td>
</tr>
<tr>
<td>256</td>
<td>15.12</td>
<td>2.61</td>
<td>253</td>
<td>0.61</td>
<td>2.83</td>
<td>2.61</td>
</tr>
<tr>
<td>512</td>
<td>12.38</td>
<td>2.24</td>
<td>253</td>
<td>0.42</td>
<td>2.47</td>
<td>2.24</td>
</tr>
</tbody>
</table>

Comparison for AAS with MSPAI and UMFpack for the NS problem, 48K dofs
3rd Level

MPI

OpenMP

MPI

OpenMP
Threaded Implementation

So, instead of having one MPI process per core, let’s have one per node using a threaded version.

Question: is MSPAI thread-safe?
Threaded Implementation

So, instead of having one MPI process per core, let’s have one per node using a threaded version.

Question: is MSPAI thread-safe?

Answer: yes (well, sort ...)

Monday, November 21, 2011
Threaded Implementation

The only problem was a race condition:

```c
unsigned int  *bitvec   = NULL,
*reset_vec = NULL;
```

Making those arrays local inside `Get_I_Set` did the trick.
Threaded Implementation

2 possible versions:

- #omp parallel region
- #omp task (OMP 3.0)

<table>
<thead>
<tr>
<th># threads</th>
<th>time</th>
<th>speedup</th>
<th>time task</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.60</td>
<td>1</td>
<td>23.12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11.33</td>
<td>1.81</td>
<td>13.49</td>
<td>1.71</td>
</tr>
<tr>
<td>3</td>
<td>7.83</td>
<td>2.63</td>
<td>7.89</td>
<td>2.93</td>
</tr>
<tr>
<td>4</td>
<td>6.02</td>
<td>3.42</td>
<td>5.78</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4.57</td>
<td>4.5</td>
<td>4.67</td>
<td>4.95</td>
</tr>
<tr>
<td>6</td>
<td>3.93</td>
<td>5.24</td>
<td>3.98</td>
<td>5.81</td>
</tr>
</tbody>
</table>
Test systems: Palu@CSCS Magny-Cours, 24 cores, and Sandy Bridge, 16 cores.
CUSP: Sparse Lin. Alg. for GPUs

```c
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void) {
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);
    return 0;
}
```
CUSP: Sparse Lin. Alg. for GPUs

- CUda SParse: a highly templated library for GPUs and CPUs, providing a high level interface that hides GPU complexities

```cpp
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```
CUSP: Sparse Lin. Alg. for GPUs

- C Ud a SP arse: a highly templated library for GPUs and CPUs, providing a high level interface that hides GPU complexities

- Uses Thrust, a C++ standard template library for GPUs

```cpp
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```
Goals of student project

- CUSP implementation of SPAI for a fixed sparsity pattern
- Evaluate CUSP implementation of the QR factorization
- Incorporate SPAI into GMRES solver implemented in another student project
- Evaluate GPU performance of optimized SPAI

```cpp
std::cout << "Solving with SPAI preconditioner" << std::endl;
// allocate storage for solution (x) and right hand side (b)
cusp::array1d<ValueType, MemorySpace> x(A.num_rows, 0);
cusp::array1d<ValueType, MemorySpace> b(A.num_rows, 1);
    // set stopping criteria (iteration_limit = 1000, relative_tolerance = 1e-6)
cusp::default_monitor<ValueType> monitor(b, 1000, 1e-6)
// setup preconditioner
    cusp::csr_matrix<IndexType, ValueType, MemorySpace> I;
cusp::gallery::eye(I, A.num_rows, A.num_cols);
cusp::precond::spai<ValueType, MemorySpace> M(A, A);
// solve
    cusp::krylov::bicgstab(A, x, b, monitor, M);
```
Fixed Sparsity SPAI

for each column $k$ from 1 to $n$

find sets of row and column indices

build $\hat{A}(k)$ matrix

do QR decomposition of $\hat{A}(k)$

compute $\hat{\mathcal{M}}$

put $\hat{\mathcal{M}}$ into preconditioner matrix

end for

$$\mathcal{J} \leftarrow \text{set of indices } j \text{ such that } m_k(j) \neq 0$$

let $\hat{m}_k$ represent the reduced vector $m_k(\mathcal{J})$

$$\mathcal{I} \leftarrow \text{set of indices } i \text{ such that } A(i, \mathcal{J}) \text{ is not identically zero}$$

$$\hat{A} \leftarrow \text{submatrix } A(\mathcal{I}, \mathcal{J})$$

$$Q \begin{pmatrix} R \\ 0 \end{pmatrix} \leftarrow qr(\hat{A}), \text{ where } R \text{ has size } n \times n$$

$$\hat{c} \leftarrow Q^T \hat{e}_k$$

$$\hat{m}_k \leftarrow R^{-1} \hat{c}(1 : n)$$

get sizes

get row indices

thrust::sort

thrust::unique

populate_Ahats

qr

QR decomposition of $\hat{A}$

get number of columns in $\hat{A}$ and an upper bound for the number of rows

get all the indices of the rows in $A$ which must land in $\hat{A}$ (with duplicates)

get set of indices of the rows which must be taken from $A$ and update the size of $\hat{A}$

with the right number of rows

copy values from $A$ to $\hat{A}$

make_mhats

solve $R \hat{m} = Q(:, k)$ using backward substitution
QR implemented in CUSP

- QR performs well on GPU only for large matrices
- Still an order of magnitude slower than MAGMA
- SPAI requires many independent small QR factorizations
Memory Allocation Scheme

- Place a large number of $\hat{A}$ into a contiguous buffer
- Use a single thread block for each matrix
Present work

- First QR implementation calculated full Q, not just a row => memory problems. *Now fixed.*

- First implementation put all $\hat{A}$ into buffer => memory problems. Solution: performance plateaus at roughly 30 $\hat{A}$; use fixed size buffers, add only the as many $\hat{A}$ as will fit in the buffer. *Implementation ongoing.*

- Use MAGMA QR... BUT: MAGMA not currently capable of assigning thread blocks to individual matrices. *Discussion with MAGMA developers ongoing.*
Future Work

- Mix all 3 levels of parallelism:
  - 1 core for MPI communications
  - 1 core for accelerator communications
  - the rest for OpenMP

Overlap communications with computation:

- fetching rows while treating local rows, computing QR or LU ...
- use accelerator to compute all local QR/LU factorization
- any idea?
Future Work

- Have the GPU use a non-constant sparsity pattern,
- Use the GPU to compute QR factorizations by “glueing” Ahat matrices together,
- Use cashing, hashing and probing ...
Future Work

More general work:

- improve classes, templating, derivation, abstraction design in general,
- use of Trilinos matrix class directly to avoid transformation cost?
- improve MSPAI as a general preconditioner,
- use MSPAI as a smoother for the ML package.
Thank You!

Any questions?...