Performance Optimization of ESMF

Gerhard Theurich, SGI
ESMF Core Team
12/17/2008
ESMF Performance?

- ESMF is a very large library.
- Some areas are more performance critical than others.
- Coupling between massively parallel components is at the core of ESMF.
- Expect most of the runtime of ESMF applications in user code, however ESMF not to become bottle neck!
ESMF_Array class (ESMF_Field)  
user data in index space

- I4, I8, R4, R8
- 1D – 7D
- ESMF or user allocated memory.
Sparse Matrix Multiplication

basic comm kernel for: Redist(), Regrid(), Halo()

\[ \text{dstArray}(i) = \sum_j \text{factor}(i,j) \times \text{srcArray}(j) \]
Sparse Matrix Multiplication

*Features*

- Reuse of precomputed communication pattern for congruent, but different (Array, Field) objects.
- Support bundles (Array, Field)
- Mixed type (I4, I8, R4, R8) support for source, destination and factors (implicit type casting).
- Zero-out of none, all, or only-touched destination elements.
- Support factors to be supplied in parallel.
- Support execution from within Components (Direct Coupling).
Sparse Matrix Multiplication

Performance

• Precomputation and Execution
  - o.k. for precomputation to “pay” for faster execution time, but there is a limit.

• Runtime and Memory footprint
  - precomputed communication pattern as small and as fast as possible
  - keep memory use during precomputation under control

• Scaling
  - weak scaling expected
  - strong scaling with limit for very high processor counts
  - prevent increase in runtime for high processor counts
Sparse Matrix Multiplication

**Performance**

- **Portability challenges**
  - machine and problem specifics may affect choice of parameters, choice of operations, operation ordering, ...
  - architectures come and go, e.g. Cray X1, ia64, ...

- **Peta-scale challenges**
  - peta-scale means massively parallel giga-scale cores
    - sequential parts of algorithms stay in giga FLOP range
  - terms that did not matter for \( P<1000 \) now do matter for runtime as well as memory footprint
  - multi-core systems shift balance between subsystems
    - memory bandwidth bottleneck even more challenging
Sparse Matrix Multiplication

*eXtreme eXchange Engine (XXE)*

- Common facility to encode parallel data exchanges and data manipulations.
- Extremely low overhead during execution.
- Partial execution option to support self-tuning during precomputation step.
- Profiling feature to help optimization.
- Highly tuned low level operations with architecture specific alternatives.
- Extensible – easy to add new operations.
- Low level automated optimization.
Sparse Matrix Multiplication

*Implementation Details*

- Use of distributed directory pattern (Pinar) to store and process sparse factor matrix in parallel.
  - efficient memory use
  - good scaling characteristics
- Optimize XXE stream:
  - staggered comm. pattern to lower contention (Roweth)
  - find optimal pipelining depth for overlap (Iancu)
  - balance source and destination work-loads
  - allow vectorization when possible
# Performance Benchmarks

## 2D Array Test Configurations

<table>
<thead>
<tr>
<th>#processors</th>
<th>Source Decomp</th>
<th>Block Size</th>
<th>Destination Decomp</th>
<th>Block Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>44x1</td>
<td></td>
<td>180x360</td>
<td>2x2</td>
<td>540x320</td>
</tr>
<tr>
<td>8x4</td>
<td></td>
<td>360x90</td>
<td>4x2</td>
<td>270x320</td>
</tr>
<tr>
<td>168x2</td>
<td></td>
<td>90x180</td>
<td>4x4</td>
<td>270x160</td>
</tr>
<tr>
<td>324x8</td>
<td></td>
<td>180x45</td>
<td>8x4</td>
<td>135x160</td>
</tr>
<tr>
<td>6416x4</td>
<td></td>
<td>45x90</td>
<td>8x8</td>
<td>135x80</td>
</tr>
<tr>
<td>1288x16</td>
<td></td>
<td>90x23</td>
<td>16x8</td>
<td>68x80</td>
</tr>
<tr>
<td>25632x8</td>
<td></td>
<td>23x45</td>
<td>16x16</td>
<td>68x40</td>
</tr>
<tr>
<td>51216x32</td>
<td></td>
<td>45x12</td>
<td>32x16</td>
<td>34x40</td>
</tr>
<tr>
<td>102464x16</td>
<td></td>
<td>12x23</td>
<td>32x32</td>
<td>34x20</td>
</tr>
<tr>
<td>204832x64</td>
<td></td>
<td>23x6</td>
<td>64x32</td>
<td>17x20</td>
</tr>
<tr>
<td>4096128x32</td>
<td></td>
<td>6x12</td>
<td>64x64</td>
<td>17x10</td>
</tr>
<tr>
<td>819264x128</td>
<td></td>
<td>12x3</td>
<td>128x64</td>
<td>9x10</td>
</tr>
<tr>
<td>16384256x64</td>
<td></td>
<td>3x6</td>
<td>128x128</td>
<td>9x5</td>
</tr>
</tbody>
</table>

2D Array

ArrayBundle of 2D Arrays

(1+2)D = 3D Array

benchmark program and configuration courtesy Peggy Li, JPL
Performance Benchmarks

**sparseMatMul() runtime**

some data courtesy Peggy Li, JPL
Performance Benchmarks

\textit{\texttt{sparseMatMul}}() \textit{runtime}

---

data courtesy Peggy Li, JPL
Performance Benchmarks

sparseMatMulStore() runtime

data courtesy Peggy Li, JPL
Performance Benchmarks

sparseMatMul() self-tuning

2D Array with balanced src/dst block distribution

SGI Altix ICE, Pleiades, 8cores/node
Performance Benchmarks

\textit{sparseMatMul()} self-tuning

2D Array with unbalanced static (4x1) src block distribution

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Performance comparison of self-tuned and configured versions of \textit{sparseMatMul} with SGI Altix ICE and Pleiades, 8 cores per node.}
\end{figure}
Performance Benchmarks

`sparseMatMul()` self-tuning

2D Array with unbalanced static (4x1) src block distribution

Cray XT4, Jaguar, 4cores/node
Future

- Further reduce `sparseMatMulStore()` overhead, scaling behavior.
- Halo() support.
- Non-blocking `sparseMatMul()`.
- DE non-blocking paradigm.
References


• Iancu: Costin Iancu and Erich Strohmaier: “Optimizing Communication Overlap for High-Speed Networks”, PPoPP '07 March 14-17, 2007

• Roweth: D. Roweth and A. Moody: “Performance of All-to-All on QsNetII” - report at http://www.quadrics.net/