Energy Consumption Evaluation for Krylov Methods on a Cluster of GPU Accelerators

Serge G. Petiton\textsuperscript{a} and Langshi Chen\textsuperscript{b}

April the 6\textsuperscript{th}, 2016

\textsuperscript{a} Université de Lille, Sciences et Technologies

and

CNRS, Maison de la Simulation, Paris-Saclay, FRANCE

\textsuperscript{b} School of Informatics and Computing

Indiana University Bloomington, USA

This work was supported by the HPC Centre Champagne-Ardenne ROMEO
Outline

• Introduction
• Krylov Method, GMRES as an example
• Energy consumption evaluation for Krylov methods
• Conclusion
Outline

• Introduction
• Krylov Method, GMRES as an example
• Energy consumption evaluation for Krylov methods
• Conclusion
With new programming paradigms and languages, extreme computing (exascale and beyond) would have to face several critical problems, such as the following:

- Minimize the global computing time,
- Accelerate the convergence, use a good preconditionner
- Numerical stability has to be maintained (at least)
- Minimize the number of communications (optimized Ax, asynchronous comp, communication compiler and mapper,....)
- Minimize the number of longer size scalar products,
- Minimize memory space, cache optimization....
- Select the best sparse matrix compressed format,
- Mixed arithmetic
- **Minimize energy consumption**
- ....

The goal of this talk is to illustrate that we would need “smart” auto-tuning of several parameters to **minimize the computing time and the energy consumption** for intelligent linear algebra methods to create the next generation of *High Performance Numerical software for Extreme Computing*
Outline

• Introduction
• Krylov Method, GMRES as an example
• Energy consumption evaluation for Krylov methods
• Conclusion
GMRES example: about memory space, dot products and sparse matrix-vector multiplication

Putting $x_k$ is now replaced by the minimizer of $\phi(y)$. Hence we do not need to solve the system $Ax = b$.

Algorithm 3: The generalized minimal residual method (GMRES)

1. **Start**: Choose $x_0$ and compute $r_0 = f - Ax_0$ and $v_1 = r_0 / \| r_0 \|$
2. **Iterate**: For $j = 1, 2, \ldots, k, \ldots$, until satisfied do:
   - $h_{i,j} = (Av_j, v_i), i = 1, 2, \ldots, j$.
   - $\hat{v}_{j+1} = Av_j - \sum_{i=1}^j h_{i,j}v_i$.
   - $h_{j+1,j} = \| \hat{v}_{j+1} \|$, and
   - $v_{j+1} = \hat{v}_{j+1} / h_{j+1,j}$.
3. **Form the approximate solution**:
   - $x_k = x_0 + V_k y_k$, where $y_k$ minimizes (7).

Memory space:
- sparse matrix: nnz elements
- Krylov basis vectors: $n \times m$
- Hessenberg matrix: $m \times m$

Scalar products, at $j$ fixed:
- Sparse Matrix-vector product: $n$ of size $C$
- Orthogonalization: $j$ of size $n$

$m$, the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, with better or approximately same convergence behaviors.
GMRES example: about memory space, dot products and sparse matrix-vector multiplication

\( \mathbf{x}_k \) is now replaced by the minimizer of \( \varphi(y) \). Hence we do not have to store all the Krylov basis vectors.

Algorithm 3: The generalized minimal residual method

1. **Start:** Choose \( \mathbf{x}_0 \) and compute \( \mathbf{r}_0 = \mathbf{f} - \mathbf{A}\mathbf{x}_0 \) and \( \mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\| \).
2. **Iterate:** For \( j = 1, 2, \cdots, k, \cdots \), until satisfied do:
   
   \[
   h_{i,j} = (\mathbf{A}\mathbf{v}_j, \mathbf{v}_i), \quad i = 1, 2, \cdots, j, \\
   \hat{\mathbf{v}}_{j+1} = \mathbf{A}\mathbf{v}_j - \sum_{i=1}^{j} h_{i,j}\mathbf{v}_i, \\
   h_{j+1,j} = \|\hat{\mathbf{v}}_{j+1}\|, \text{ and} \\
   \mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1}/h_{j+1,j}.
   \]
3. **Form the approximate solution:**
   \[
   \mathbf{x}_k = \mathbf{x}_0 + \mathbf{V}_k\mathbf{y}_k, \text{ where } \mathbf{y}_k \text{ minimizes (7).}
   \]

Memory space:
- sparse matrix: \( \text{nnz} \) elements
- Krylov basis vectors: \( n \times m \)
- Hessenberg matrix: \( m \times m \)

Scalar products, at \( j \) fixed:
- Sparse Matrix-vector product: \( n \) of size \( C \)
- Orthogonalization: \( j \) of size \( n \)

\( m \), the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, with better or approximately same convergence behaviors.
GMRES : about memory space and dot products

GMRES is now replaced by the minimizer of $f(y)$. Hence we describe the method.

**Algorithm 3:** The generalized minimal residual method (GM).

1. **Start:** Choose $x_0$ and compute $r_0 = f - Ax_0$ and $v_1 = r_0 / \| r_0 \|.$
2. **Iterate:** For $j = 1, 2, \cdots, k, \cdots$, until satisfied do:
   
   $h_{i,j} = (Av_j, v_i)$, $i = 1, 2, \cdots, j$,
   
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j}v_i$,
   
   $h_{j+1,j} = \| \hat{v}_{j+1} \|$, and
   
   $v_{j+1} = \hat{v}_{j+1} / h_{j+1,j}$.
3. **Form the approximate solution:**
   
   $x_k = x_0 + V_k y_k$, where $y_k$ minimizes (7).

**Incomplete orthogonalization (Y. Saad):** i.e. i.e. $i= \text{from}$

max(1, j-q) to j

$q>0$. Then, $J-q+1$ bands on the Hesseberg matrix.

Memory space:

- sparse matrix: nnz (i.e. < C n) elements
- Krylov basis vectors: n m
- Hessenberg matrix: m m

$m$, the subspace size, may be auto-tuned at runtime to minimize the space memory occupation and the number of scalar product, with better or approximately same convergence behaviors. The number of vectors orthogonalized with the new one may be auto-tuned at runtime. The subspace size may be large!
GMRES: about memory space and dot products

Solving $x_k$ is now replaced by the minimizer of $F(y)$. Hence we describe the method.

**Algorithm 3:** The generalized minimal residual method (GMRES)
1. **Start:** Choose $x_0$ and compute $r_0 = f - Ax_0$ and $v_1 = r_0 / \|r_0\|$. 
2. **Iterate:** For $j = 1, 2, \ldots, k, \ldots$, until satisfied do:
   \[
   h_{i,j} = (Av_j, v_i), \quad i = 1, 2, \ldots, j, \\
   \hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i, \\
   h_{j+1,j} = \|\hat{v}_{j+1}\|, \quad \text{and} \\
   v_{j+1} = \hat{v}_{j+1} / h_{j+1,j}.
   \]
3. **Form the approximate solution:** 
   \[x_k = x_0 + V_k y_k, \quad \text{where } y_k \text{ minimizes (7)}.\]

**Memory space:**
- Sparse matrix: nnz (i.e., $< Cn$) elements
- Krylov basis vectors: $n \times m$
- Hessenberg matrix: $m \times m$

**Scalar products, at $j$ fixed:**
- Sparse Matrix-vector product: $n$ of size $C$
- Orthogonalization: $m$ of size $n$

[see papers with P.-Y. Aquilanti (TOTAL) and T. Katigari (U. Tokyo)]

**Other technique:** so-called “Communication Avoiding” (CA): we first compute a non-orthogonal basis +TSQR to orthogonalize the vectors
GMRES: about memory space and dot products

Putting \( x_k \) is now replaced by the minimizer of \( g(y) \). Hence we describe the method.

Algorithm 3: The generalized minimal residual method (GMRES)
1. **Start**: Choose \( x_0 \) and compute \( r_0 = f - Ax_0 \) and \( v_1 = r_0 / \| r_0 \| \).
2. **Iterate**: For \( j = 1, 2, \ldots, k, \ldots \), until satisfied do:
   \[ h_{i,j} = (Av_j, v_i), \ i = 1, 2, \ldots, j, \]
   \[ \hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j}v_i, \]
   \[ h_{j+1,j} = \| \hat{v}_{j+1} \|, \text{ and} \]
   \[ v_{j+1} = \hat{v}_{j+1} / h_{j+1,j}. \]
3. **Form the approximate solution**:
   \[ x_k = x_0 + V_k y_k, \text{ where } y_k \text{ minimizes (7)}. \]

Memory space:
- Sparse matrix: \( \text{nnz (i.e. } < C \ n) \text{ elements} \)
- Krylov basis vectors: \( n \ m \)
- Hessenberg matrix: \( m \ m \)

Scalar products, at \( j \) fixed:
- Sparse Matrix-vector product: \( n \) of size \( C \)
- Orthogonalization: \( m \) of size \( n \)

[see papers with P.-Y. Aquilanti (TOTAL) and T. Katigari (U. Tokyo)]

Other technique: so-called “Communication Avoiding” (CA): we first compute a non-orthogonal basis +TSQR to orthogonalize the vectors

What about the energy consumption with respect to these different versions? Is the faster version the more energy efficient? Does exist an unique solution to minimize all these criteria (computing time, energy, memory space..)
Testing Methods and Matrices

Different Orthogonalizations

Arnoldi (abbr AR)
- Standard Arnoldi Orthogonalization (CGS)

ArnoldiHG (abbr ARHG)
- Standard Arnoldi + Hypergraph SpMV

IArnoldi(q) (abbr IAR)
- Incomplete Arnoldi Orthogonalization

IArnoldiHG(q) (abbr IARHG)
- Incomplete Arnoldi + Hypergraph SpMV

CDia, EDia have scalable size; FullChip has $n = 2987012, nnz = 26621983$; HV15R has $n = 2017169, nnz = 283073458$. 
Outline

• Introduction
• Krylov Method, GMRES as an example
• Energy consumption evaluation for Krylov methods
• Conclusion
Powmon/2.0 within the Cluster ROMEO

Impact of Powmon/2.0 on Execution time (s) of MPI program

<table>
<thead>
<tr>
<th>MPI process</th>
<th>2</th>
<th>12</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>No measure</td>
<td>7.075806</td>
<td>7.074594</td>
<td>7.075586</td>
<td>7.079222</td>
</tr>
<tr>
<td>CPU measure (0ms)</td>
<td>9.288037</td>
<td>14.080540</td>
<td>19.853754</td>
<td>18.494354</td>
</tr>
<tr>
<td>CPU measure (1ms)</td>
<td>7.087719</td>
<td>7.084694</td>
<td>7.111734</td>
<td>7.097556</td>
</tr>
<tr>
<td>CPU measure (10ms)</td>
<td>7.082563</td>
<td>7.082486</td>
<td>7.086433</td>
<td>7.107893</td>
</tr>
<tr>
<td>GPU measure (0ms)</td>
<td>7.244987</td>
<td>7.101177</td>
<td>7.280451</td>
<td>7.585576</td>
</tr>
<tr>
<td>GPU measure (1ms)</td>
<td>7.086481</td>
<td>7.085908</td>
<td>7.188792</td>
<td>7.284488</td>
</tr>
<tr>
<td>GPU measure (10ms)</td>
<td>7.089824</td>
<td>7.080087</td>
<td>7.139392</td>
<td>7.187987</td>
</tr>
</tbody>
</table>

- In-band sensors updates status every millisecond
- We query in-band sensors every 10 milliseconds.

The cluster ROMEO

A cluster of ROMEO HPC Center in Champagne-Ardenne. It has 130 node, each is made of 2 Xeon E5 CPU and 2 NVIDIA K20x GPUs. It has a total 254.9 TFlops and ranks #279 in Top 500 June 15, #14 in Green 500 June 15
Instantaneous Power of GPU

Instantaneous Power (Watt) = power value of in-band sensors

- start, computation, terminate stages
  - Start: initialization
  - Computation: SpMV, inner product, communication
  - Terminate: measure basis orthogonality (MM product)
- ArnoldiHG has a power higher than Arnoldi
- Arnoldi has a power higher than IArnoldi(q)
- PCAQRmQE has higher power than Arnoldi.
Instantaneous Power of GPU

Instantaneous Power (Watt) = power value of in-band sensors

- start, computation, terminate stages
  - Start: initialization
  - Computation: SpMV, inner product, communication
  - Terminate: measure basis orthogonality (MM product)
- ArnoldiHG has a power higher than Arnoldi
- Arnoldi has a power higher than lArnoldi(q)
- PCAQRmQE has higher power than Arnoldi.
Instantaneous Power of GPU

Instantaneous Power (Watt) = power value of in-band sensors

- start, computation, terminate stages
  - Start: initialization
  - Computation: SpMV, inner product, communication
  - Terminate: measure basis orthogonality (MM product)
- ArnoldiHG has a power higher than Arnoldi
- Arnoldi has a power higher than IArnoldi(q)
- PCAQRmQE has higher power than Arnoldi.
Instantaneous Power of CPU

Instantaneous Power (Watt) \( = \) power value of in-band sensors

- Start stage: high power (Initialization)
- Computation, Terminate stage: low power (Communication)
Instantaneous Power of RAM

Instantaneous Power (Watt) = power value of in-band sensors

- Start stage: high power (Initialization)
- Computation, Terminate stage: low power (Communication)
- Fluctuation: imbalance of data transfer
Total Energy Consumption (TEC)

$\text{TEC (Joule)} = \text{Energy of GPU} + \text{Energy of CPU} + \text{Energy of RAM}$ (others like overhead of cooling is not considered by us)

- hypergraph saves TEC in Arnoldi
- Truncation of Arnoldi has no significant benefit of saving TEC
- PCAQRmQE consumes much more TEC than Arnoldi
- Matrix structure affects the saving of TEC
Scalability of TEC

- ArnoldiHG and PCAQRmQE have good scalability of TEC
- Communication is important to scalability of energy consumption.
Total Energy Consumption

GMRES

- PCAQR-GMRES consumes more energy than ARHG-GMRES and IARHG-GMRES
- When ARHG-GMRES converges, it consumes the least energy.
Outline

• Introduction
• Krylov Method, GMRES as an example
• Energy consumption evaluation for Krylov methods
• Conclusion
Conclusion

- We have to find a tradeoff between several minimizations (time of each iteration, global time to convergence (i.e. number of iteration), accuracy, memory space, cache utilizations, and energy consumption)
- Optimizing some parameters from architecture, numerical method, algorithm, parallelism, memory space, multi-core utilizations,…… would not lead to an unique solution
- End users would have to decide what criteria to minimize
- Expertise from end-users would be exploited through new high level language and/or framework (YML, PGAS,……) – cf. yml.prism.uvsq.fr
- We have to analyse auto-tuned numerical methods to find new crieteria to evaluate the quality of the converge and to decide actions

*The method may decide to compute others parameters just to take the decision, and they may learnt (linear algebra learning?) leading to intelligent linear algebra.*