Asynchronous iterative algorithms & Reliability

Mourad Hakem

LIFC Laboratory - University of Franche Comté - France

AND Team

Workshop Inria - Illinois 2009 - Paris

Juin 11, 2009
1 Introduction

2 Iterative computing model

3 Reliability

4 Conclusion
Motivation

- Solve large linear / non-linear equation systems
- Result in problems composed of millions of components
- Single computing unit? — Parallel and distributed platforms
Direct and iterative resolution methods

Equation systems can be solved with direct and iterative methods.

Direct method

- Gives the **exact solution** for a problem after executing a finite number of operations.
- Does not solve all kinds of problems.
- Not well adapted for large problems.

Iterative method

- Iterates until it gives a **good approximation** of the solution.
- Sometimes it is the only way to solve a problem.
- Well adapted for large problems and easy to parallelize.
Iterative methods: Synchronous vs Asynchronous iteration model

Synchronous iteration model

- Processor 1
- Processor 2
- Time

Asynchronous iteration model

- Processor 1
- Processor 2
- Time
Asynchronous iteration model

Advantages

- Eliminates idle times.
- Eliminates synchronizations.
- Tolerates message loss.
- Adapted for heterogeneous and volatile environments.

Difficulties

- Requires a specified asynchronous messaging mechanism.
- Hard to detect the global convergence.

All the difficulties can be overcome with a dedicated platforms:

JACE and JACEP2P-V2
Iterative model

To solve $Ax = b$,

Split $A = M - N$, $M$ nonsingular

Sequential case:

\[
\begin{align*}
\text{given } x^0 \\
k = 1, 2, \ldots
\end{align*}
\]

\[
x^{k+1} = M^{-1}Nx^k + M^{-1}b
\]

\[
\equiv x^{k+1} = F(x^k)
\]
Asynchronous iteration model

The asynchronous model is defined by

\[
\begin{align*}
\text{Given } X^0 &= (X_1^0, \ldots, X_m^0) \\
&\text{for } k = 1, 2, \ldots \\
&\text{for } i = 1, \ldots, m \\
X_i^{k+1} &= \begin{cases} 
F_i(X_1^{r_i^1(k)}, \ldots, X_m^{r_i^m(k)}) & \text{if } i \in S(k) \\
X_i^k & \text{if } i \notin S(k)
\end{cases}
\end{align*}
\]  

1. $S(k)$ is the set of components to be updated at step $k$
2. $k - r_j^i(k)$ is the delay of the $j^{th}$ proc.
   The $i$ proc. computing the $i^{th}$ block at the $k^{th}$ iteration
3. If $r_j^i(k) = k$ then: synchronous algorithms.

\[
F_i\left(X_1^{r_i^1(k)}, \ldots, X_m^{r_i^m(k)}\right)
\]
Convergence conditions

Theoretical results

1. \( \lim_{k \to \infty} (M^{-1}N)^k = 0 \iff \rho(M^{-1}N) < 1 \)

2. Synchronous iterations:
   \( \rho(M^{-1}N) < 1 \implies (1) \text{ converges to } A^{-1}b \)

3. Asynchronous iterations:
   \( \rho(|M^{-1}N|) < 1 \implies (2) \text{ converges to } A^{-1}b \)
Motivation

Context
- Heterogeneous platforms - Clusters and Grids
- Iterative applications

Failures?
- Software is assumed to be reliable
- Only hardware failures of nodes
- Faults are assumed to be fail-silent/fail-stop

Fault tolerance objective?
tolerate at most $r$ (reliability factor) node failures
Problem and solutions

Bi-criteria problem

\textit{latency & reliability : antagonist objectives}

Solutions:

- checkpointing
- task replication

  - Primary/Backup (passive replication)
    - backup is activated only if the fault occurs
    - requires \textit{fault detection/recovery mechanism}

  - Active replication
    - all replicas are activated in parallel
    - no \textit{fault detection/recovery mechanism}
**Problem and solutions**

**Bi-criteria problem**

*latency & reliability: antagonist objectives*

Solutions:

- checkpointing
- task replication

  - Primary/Backup (passive replication)
    - backup is activated only if the fault occurs
    - requires *fault detection/recovery mechanism*

  - Active replication
    - all replicas are activated in parallel
    - no *fault detection/recovery mechanism*
Increasing reliability

- Without replication
  \[ R_A = \prod_{i=1}^{n} R_i \]

- With replication
  \[ R_A = \prod_i R_i = \prod_i \left( 1 - \prod_j (1 - R_j) \right), \quad 1 \leq i \leq n, \quad 1 \leq j \leq r+1 \]
A brief description of FT-Jace environment

FT-Jace: Reliable Parallel Programming Model for Distributed Computing Environments

**Principle**
- Multi-threaded
- Convergence detection - Asynchronous iteration model
- Uses the active software *replication scheme* to *mask failures*
- Can tolerate a given number $r$ of arbitrary node failures
A brief description of FT-Jace environment

FT-Jace: Reliable Parallel Programming Model for Distributed Computing Environments

Principle

- Multi-threaded
- Convergence detection - Asynchronous iteration model
- Uses the active software replication scheme to mask failures
- Can tolerate a given number $r$ of arbitrary node failures
Communication overhead reduction

**Communication overhead:** $e \rightarrow e(r + 1)^2$

- Duplicating each task $r + 1$ times is an **absolute requirement**

- But duplicating each communication $(r + 1)^2$ times is **not mandatory**

**How to reduce communication overhead?**
Communication overhead reduction

**All to ALL → One To One mapping?**
Idea: Try to decrease communication overhead from $e(r + 1)^2$ down to at most $e(r + 1)$

**Property 1:**
Let $t$ and $t_*$ two tasks involved in communication. If a replica of task $t$ and a replica $t^*_z$ of $t_*$ are mapped on the same node $P$, then there is no need for other replicas of $t_*$ to send data to node $P$. 
Communication overhead reduction

Counter example
Communication overhead reduction

Counter example
Communication overhead reduction

Counter example
The number of communications will also be bounded by $e(r + 1)$ if at each step replicas are assigned to different processors.
The number of communications will also be bounded by $e(r + 1)$ if at each step replicas are assigned to different processors.
Some theoretical results

(Joint work with A. Benoit and Y. Robert)

- Fork/Outforest graph applications
  \[ e(r + 1) \]

- Classical kernels/Parallel algorithms: LU, LAPLACE, STENCIL, DOOLITTLE, LDMt
  \[ V_2(r + 1) + V_3 \left( r \left\lceil \frac{r+2}{2} \right\rceil + 2 \right), \quad V_2 \leq \left\lfloor \frac{e}{2} \right\rfloor, \quad V_3 \leq \left\lfloor \frac{e}{3} \right\rfloor \]

- General graph applications:
  \[ e \left( r \left\lceil \frac{r+2}{2} \right\rceil + 1 \right) \]
Experimental results

Aim
- Evaluation of FT-Jace performance
- Comparison with fault-free version \((r = 0)\)

Parameters
- FreSh Grid’5000 platform
- 50 nodes, bi-core Opteron 2.0 GHZ – Gigabit Ethernet
- Jacobi method to solve linear systems \((Ax = b)\)
  - \((\text{Synchronous & Asynchronous mode})\)
- size of the matrix \(A\) is \(10^8 \times 10^8\).
- \(r = \{1, 2, \ldots, 10\}\)

Metrics
- Latency
- Overhead
Experimental results

The achieved latency

Asynchronous mode

Synchronous mode

![Graph showing execution time vs reliability factor for asynchronous and synchronous modes.](image-url)
Experimental results

Table: Fault tolerance overhead (%)

<table>
<thead>
<tr>
<th>Reliability factor ($r$)</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous mode</td>
<td>0.04</td>
<td>0.06</td>
<td>0.17</td>
<td>0.30</td>
<td>0.56</td>
</tr>
<tr>
<td>Synchronous mode</td>
<td>0.06</td>
<td>0.16</td>
<td>0.52</td>
<td>0.69</td>
<td>1.06</td>
</tr>
</tbody>
</table>

Fault tolerance overhead increases slightly as reliability factor goes up.
Conclusion

- Iterative computing model
- Asynchronous model is reliable by nature
- Reliability: **FT-Jace** environment
- Good performance

Perspective

- Effective scheduling strategy
- Checkpointing?