Asynchronous iterative algorithms for computational science on the grid: three case studies

Jacques M. Bahi – Raphaël Coutilier – Philippe Vuillemin

{bahi,coutilier,vuillemi}@iut-bm.univ-fcomte.fr.

Laboratoire d’Informatique de l’université de Franche-Comté
Introduction

Context: both heterogeneous machines and networks -> possible low performance if lots of communications.

Iterative methods: approximate results at each iteration (synchronously or asynchronously).

Parallel iterative algorithms implemented with Jace, a Java based library optimized for distributed iterative asynchronous computation. (http://info.iut-bm.univ-fcomte.fr/staff/mazouzi/jace.html)
1. Parallel iterative algorithms
   1.1. Classification
   1.2. Conclusions about asynchronism

2. Jace: Java Asynchronous Computation Environment
   2.1. Presentation of Jace
   2.2. Parallelization methodology

3. Presentation of the problems
   3.1. Linear problem: Ax = b
   3.2. Nonlinear differential equation problems

4. Experiments

Conclusion

Asynchronous iterative algorithms for computational science on the grid: three case studies – p.3/??
1. Parallel iterative algorithms

1.1. Classification

- Synchronous Iterations, Synchronous Communications (SISC)

Asynchronous iterative algorithms for computational science on the grid: three case studies – p.4/??
1. Parallel iterative algorithms

Synchronous Iterations, Asynchronous Communications (SIAC)

Communications overlapped by computations -> less idle times.
1. Parallel iterative algorithms

Asynchronous Iterations, Asynchronous Communications (AIAC)

No synchronisation between two iterations -> no idle time.

Asynchronous iterative algorithms for computational science on the grid: three case studies – p.6/??
1. Parallel iterative algorithms

1.2. Conclusions about asynchronism

- Number of iterations greater,
- Warning: ensure convergence!

But:

- All idle times suppressed,
- Execution time considerably reduced, especially in distant cluster context,
- Tolerant to long message delays,
- Message loss tolerant,
- Efficient and easy to implement.
2. JACE

2.1. Presentation of JACE

- Java based library to implement asynchronous algorithms.
- Communications implemented with RMI (Remote Method Invocation).
- Threads to manage computation and communications asynchronously.
- Non-blocking sendings.
- Blocking receptions for synchronism and non-blocking receptions for asynchronism.
- Messages replaced in non-blocking mode.
2.2. Parallelization methodology

Always the same skeleton whatever the version:

- Initialize the dependencies
- repeat
  - Computation of local data
  - Exchange of nonlocal data
  - Convergence detection
- until Global convergence is achieved

Global convergence achieved when all local residual errors are inferior to a fixed threshold:

\[ \max_i |y_i^{t+1} - y_i^t| < \epsilon \]
Convergence detection: centralizing the state of all nodes over one (the master).

**Synchronous version:**
- Send error to master after an iteration and wait for orders (idle times),
- Master detects if all errors inferior to $\epsilon$.

**Asynchronous version:**
- Send a message to master if local convergence state changes (no idle time),
- Master tests if all nodes in convergence state.

If CONVERGENCE, master orders to stop iterations.
2. JACE

**Step 1** - Parallelizing a sequential algorithm to obtain a synchronous iterative algorithm:
- Split the data,
- Compute dependencies,
- Manage communications.

**Step 2** - Desynchronizing a synchronous algorithm to obtain an asynchronous iterative algorithm:
- Use non-blocking receptions,
- Modify convergence detection.
3. Presentation of the problems

3.1. Linear problem: $Ax = b$

- Let $A = M - N$ and $T = M^{-1}N$.
- Iterative formula: $x^{k+1} = Tx^k + M^{-1}b$.

- $M$ is block diagonal -> Block Jacobi Method.
- Asynchronous convergence criterion: $\rho(|T|) < 1$ with $\rho(T) = \max_i \{|\lambda_i|\}$ and $\lambda_i$ eigenvalue of $T$.
- $A$ is randomly generated so that $T$ is non-negative and either diagonally dominant or M-matrix.
3. Presentation of the problems

3.2. Nonlinear differential equation problems

- Problems: PDEs (Partial Differential Equations).
- Finite difference method to approximate the space derivatives.
- Implicit Euler Method to approximate the time derivative.
- Multisplitting Newton Method to solve the nonlinear system obtained.
- Remark: resulting linear systems solved with the GMRES method.
3. Presentation of the problems

One-dimensional PDE system. Equation after having approximated the space derivative:

\[ u_i' = 1 + u_i^2 v_i - 4u_i + \alpha(N + 1)^2(u_{i-1} - 2u_i + u_{i+1}) \]
\[ v_i' = 3u_i - u_i^2 v_i + \alpha(N + 1)^2(v_{i-1} - 2v_i + v_{i+1}) \]

\(u_i\) and \(v_i\) in a vector: \(y = (u_1, v_1, \ldots, u_N, v_N)\)

Dependencies for \(y_j\) with \(j = 1, \ldots, 2N\):

- Vector \(y\):
  - for \(u\) (if \(j\) is even)
    - \(j-2\) \(j-1\) \(j\) \(j+1\) \(j+2\)
  - for \(v\) (if \(j\) is odd)
    - \(j-2\) \(j-1\) \(j\) \(j+1\) \(j+2\)

Dependencies for processor \(P_l\): \(P_{l-1}\) and \(P_{l+1}\).
3. Presentation of the problems

Two-dimensional PDE system: advection and diffusion problem for 2 chemical species:

\[
\frac{\delta c^i}{\delta t} = K_h \frac{\delta^2 c^i}{\delta x^2} + V \frac{\delta c^i}{\delta x} + \frac{\delta}{\delta z} K_v(z) \frac{\delta c^i}{\delta z} + R^i(c^1, c^2, t)
\]

with:

- \( R^1(c^1, c^2, t) = -q_1 c^1 c^3 - q_2 c^1 c^2 + 2q_3(t)c^3 + q_4(t)c^2 \),
- \( R^2(c^1, c^2, t) = q_1 c^1 c^3 - q_2 c^1 c^2 - q_4(t)c^2 \),

Representation of the 2D grid into vector \( y \).

Dependencies for \( x - z \) mesh: elements North, South, East, West in the 2D spatial grid
- for processor \( P_l \): \( P_{l-1} \) and \( P_{l+1} \) (horizontal distribution of the 2D grid on processors).
4. Experiments

- Comparison of execution times in synchronous and asynchronous versions for each problem.
- 6 heterogeneous workstations (from AMD Athlon XP 1600+ to Intel Pentium 4 2.40GHz).
- Heterogeneous networks: Ethernet 100Mb/s and ADSL connexion (512/128 Kb/s).
- 1.2.2 version of the Java Development Kit (JDK).
- Linux Debian operating system.
4. Experiments

Linear problem

![Graph showing time vs matrix size for synchronous and asynchronous methods]

Asynchronous iterative algorithms for computational science on the grid: three case studies – p.17/??
4. Experiments

- **One-dimensional PDE system**

![Graph showing time versus vector size for synchronous and asynchronous algorithms.](image-url)
4. Experiments

Two-dimensional PDE system

Asynchronous iterative algorithms for computational science on the grid: three case studies – p.19/??
When ratio “computation/communication” increases, ratio “synchronous time/asynchronous time” decreases -> more processors for very large problems in AIAC mode.

AIAC mode is not penalized by synchronisations -> more efficient than synchronous mode.

Asynchronism suitable for computational science in grid context (especially iterative algorithms).

Very easy to desynchronize iterative algorithms with Jace.

Perspectives: three-dimensional PDE systems, overlapping technics, finite element methods...