Dependent Iterations in Scientific Calculations
Parallelized by the Parareal-in-Time Algorithm

Toshiya Takami, Kyushu University, Japan
In collaboration with:
K. Fukazawa, H. Honda, Y. Inadomi,
R. Susukita, T. Kobayashi, and T. Nanri

Outline

1. Introduction
   * Scientific Computing on Massively Parallel Machines
   * Dependent Calculations in Scientific Computing

2. “Parareal” Algorithm
   * Parareal-in-Time as Higher-Order Perturbation

3. How to Implement “Parareal” Calculations
   * Template class: Parareal<T>
   * Non-blocking Communications for Further Speed-up

4. Conclusion

1. Introduction

* Massively Parallel Computers
* Almost all in the Top500 List
* Large problems with many independent components
* SPMD, Master-Worker, Many Task Computing, etc.
* Dependency can be resolved by communications.
* Relatively small problems
  * Small systems, small matrices ==> Low performance
  * Sequential calculations ==> Not parallelize

Parallel Application Programs

Roughly speaking, applications are usually configured in these types.
1. Introduction

**Dependent Calculations in Scientific Computing**

- Dependent calculations: \( x_{k+1} = \mathcal{F}_k(x_k) \)
- Time evolutions in the initial value problem:
  - Simulations by explicit/implicit solvers
- Convergent sequence of state vectors:
  - Iterative calculations for linear/nonlinear equations
- Krylov subspace methods for linear problems

2. Parareal Algorithm & Applications

**What is Parareal-in-Time?**

- “Its main goal concerns real time problems, hence proposed terminology of ‘parareal’ algorithm”
- “Parareal is not the first algorithm to propose the solution of evolution problems in a time-parallel fashion”
- Multiple Shooting Methods with Newton’s Iteration
- Space-Time Multigrid Method

**Parareal Algorithm as a Newton’s Iteration (1)**

\[
\begin{align*}
  x_{k+1} &= \mathcal{F}_k(x_k) \\
  x_{k+1}^{(r+1)} &= \mathcal{G}_k(x_k^{(r+1)}) + \mathcal{F}_k(x_k^{(r)}) - \mathcal{G}_k(x_k^{(r)}) \\
  x_{k+1}^{(r+1)} &= \mathcal{G}_k(x_k^{(r+1)}) + \mathcal{F}_k(x_k^{(r)}) - \mathcal{G}_k(x_k^{(r)})
\end{align*}
\]

(Original Problem)

(Parareal-in-Time Algorithm)

- Consider the procedure as a Newton’s iteration
  - Consider a nonlinear equation
    \[
    f(X^{(r)}) = 0
    \]
    \[
    f(X^{(r)}) = \begin{bmatrix}
    s_0^{(r)} - s_0 \\
    s_1^{(r)} - s_1^{(0)} \\
    \vdots \\
    s_n^{(r)} - s_n^{(0)}
    \end{bmatrix}
    \]
  - Newton’s iteration for this problem:
    \[
    X^{(r+1)} = X^{(r)} - \left[f'(X^{(r)})\right]^{-1} f(X^{(r)})
    \]
    \[
    X^{(r+1)} = \begin{bmatrix}
    s_0^{(r+1)} \\
    s_1^{(r+1)} \\
    \vdots \\
    s_n^{(r+1)}
    \end{bmatrix}
    \]

Parareal Algorithm as a Newton’s Iteration (2)

• After some calculations, we obtain multiple shooting relations,
  
  \[ x_0^{(r)} = x_0 \]
  \[ x_{k+1}^{(r)} = \mathcal{F}_k(x_k^{(r)}) + \mathcal{F}_k'(x_k^{(r)}) \left[ x_k^{(r+1)} - x_k^{(r)} \right] \]

• Substitute the Jacobian term by the coarse solver,
  \[ \mathcal{F}_k'(x_k^{(r)}) \left[ x_{k+1}^{(r)} - x_k^{(r)} \right] \approx \mathcal{G}_k(x_k^{(r+1)}) - \mathcal{G}_k(x_k^{(r)}) \]

we obtain the Parareal-in-Time iteration

\[ x_k^{(r+1)} = \mathcal{G}_k(x_k^{(r+1)}) + \mathcal{F}_k(x_k^{(r)}) - \mathcal{G}_k(x_k^{(r)}) \]

Applications in Early Years

• Various time-evolution problems in scientific computing are parallelized by this method
  
  PDE with a fluid/structure coupling:
  
  MD with quantum interactions:
  
  Quantum optimal control problem:
  
• The coarse solver by varying the width of time-steps:

\[ x(t + \Delta t) = f_{\Delta t}(x(t)), \quad \mathcal{F}_k = [f_{\Delta t}]^n, \quad \mathcal{G}_k \equiv f_{n\Delta t} \]

Parareal Algorithm as a Higher-Order Perturbation

• ‘Perturbation’ is a basic concept in science:
  
  Unperturbed description + complicated perturbation

  \[ \text{Simple, Fast} \quad \text{Accurate but time consuming} \]

• For linear problems, we write

\[ x_{k+1} = \mathcal{F}_k x_k \]

\[ = [\mathcal{G}_k + (\mathcal{F}_k - \mathcal{G}_k)] x_k \]

• If we introduce perturbed sequence \( \{x_k^{(r)}\} \), we obtain the ‘Parareal relation’

\[ x_{k+1}^{(r+1)} = [\mathcal{G}_k + (\mathcal{F}_k - \mathcal{G}_k)] x_k^{(r+1)} \]

\[ \approx \mathcal{G}_k x_k^{(r+1)} + (\mathcal{F}_k - \mathcal{G}_k) x_k^{(r)} \]

Scientific Computing

• Expensive interactions are introduced as perturbation
  
  Molecular Dynamics with long-range interactions
  
  Quantum systems (driven by external fields)

\[ |\psi\rangle = \exp \left( \frac{\Delta t}{\hbar} H \right) |\psi\rangle = \left[ I - \left( \exp \left( \frac{\Delta t}{\hbar} H \right) - I \right) \right] |\psi\rangle \]

• Non time-evolution problems
  
  Dependent linear calculus: general linear transformations

\[ x_{k+1} = A_0 + B_k x_k \]

• Convergent sequence: Nonlinear SCF iterations

\[ FC=SCe \quad \text{(Hartree-Fock-Roothaan Equation)} \]
### Speed-up Ratio in Parallel Computing

- **Expected speed-up:** $S(r, K, P) = \frac{P}{r + TP + \frac{P(P - 1)}{K}}$
  - (dashed line)
- **Colored lines:**
  - Measured values of $S(r, K, P)$ for matrix multiplications.
  - For large problem ($N=8192$):
    - Linear speed-up
  - For small problem ($N=1024$):
    - Saturated


### 3. How to Implement “Parareal”

- This algorithm should be parallelized in relatively higher layer of the software stack.
- Useful frameworks or interfaces are required.

#### C++ template class: Parareal $<$T$>$

- To support implementation of your Parareal Program:
  - $T$ is a class to represent $x^r_k$ or $x^{(r)}_k$
- Implement methods in abstract classes:
  - PararealElement$<$T$>$
  - PararealSequence$<$T$>$
- Instantiate
  - Parareal$<$T$>$
- Use the following methods:
  - $T*$ pararealSequence(Seq, Niter)
  - $T*$ exactSequence(Seq)

```
0001 Parareal<RealVector> pararealSequence(Seq, Niter);
0002 RealVector* exactResult = SVM.exactSequence(SeqSeq);
0003 for (int i = Niter; i <= Niter; i++) {
0004   seqResult[i] = exactResult[i];
0005   std::cout << i << " " << exactResult[i].nnname() << std::endl;
0006 }
```

#### Actual Codes of Parareal$<$T$>$

- $T*$ pararealSequence(T& seq, int Niter)
  - $x^{(r+1)}_k = G_k(x^{(r+1)}_k) + F_k(x^{(r)}_k) - G_k(x^{(r)}_k)$

```
for (int r = 1; r <= Niter; r++) {
  for (int k = r; k <= Niter - r + 1; k++) {
    if (k > Nstep) break;
    seq.coarseEvolution(seq[k-1], coarse[k]);
    seq[k] = coarse[k];
    if (k == Nstep - r + 1) {
      seq[k] = modify[k];
    }
  }
}
```
Parallel Implementation

- Pipeline-type parallel executions are available, since taking barrier over the whole nodes is not necessary.

![Diagram showing parallel implementation process]

Speedup Ratio

- The ideal speedup of this algorithm is $1/T$.
- From the previous figure, the actual speedup ratio will be
  \[
  S(r, K, P) = \frac{K(T_g + T_f)}{KT_g + T_c(P-1) + rK(T_f + T_g) / P} = \frac{P}{r + TP + \frac{P(P-1)}{K}t}
  \]
- Schematic behaviour is like the right figure.

Further Speedup

- For large $P (\approx K)$, actual speedup ratios become lower than the ideal value $1/T$. \(\leqslant\) Communication time
- How can we reduce the communication time?
- It is difficult to reduce the time itself, but we can overlap the calculations and communications, which leads to recover the ideal speed-up.

Pipeline Pattern and Non-Blocking Communications

- As a future work, non-blocking communications are used to improve total performance of the Parareal-in-Time.
- "Pipeline Interface and Library" will be written by sparse-collective communications with reduction calculations.
4. Conclusion

- Through perturbative description of the “Parareal”,
  - We showed that various dependent iterations in scientific problems are in the range of the “Parareal-in-Time” method.
- Template class `Parareal<T>` for C++ is available to implement “Parareal-in-Time” calculations.
  - Since, in general, it is difficult to parallelize in an upper layer, we provide a useful library for programming.
- Future Work:
  - non-blocking communications can be used to hide the communication time and improve total performance

As the Third Scheme for Massively Parallel Computing

- As one of the typical schemes for massively parallel computing?

Thank you for your attention

- This work is supported by JST, CREST.