Portable task-based programming for Elastodynamics

Lionel BOILLOT – INRIA Magique3D, Pau (France)

Advisors:
- Hélène BARUCQ, Julien DIAZ – INRIA Magique3D, Pau
- Henri CALANDRA – TOTAL E&P, Houston

Collaborative work:
- Emmanuel AGULLO – INRIA Hiepacs, Bordeaux (France)
- George BOSILCA – ICL, University of Tennessee (USA)
Dealing with heterogeneity

Classical approach:
- MPI over CPUs
- OpenMP optimization
- CUDA over GPUs

implies:
- programming effort
- difficult to maintain
- hardware-dependent
Dealing with heterogeneity

Runtime:
- abstraction layer
- hiding heterogeneity

Scheduler:
- where to execute
- when to execute

Memory:
- handles the transfer
- guarantees consistency
Table of contents

1 MPI code implementation
2 Task-based programming
3 Numerical illustration
Outline

1 MPI code implementation

2 Task-based programming

3 Numerical illustration
Elastic wave equation (first order)

\[
\begin{align*}
\rho(\mathbf{x})\partial_t \mathbf{v}(\mathbf{x}, t) &= \nabla \cdot \mathbf{\sigma}(\mathbf{x}, t) \\
\partial_t \mathbf{\sigma}(\mathbf{x}, t) &= C(\mathbf{x}) \varepsilon(\mathbf{v}(\mathbf{x}, t))
\end{align*}
\]

Discontinuous Galerkin with Leap-frog scheme

Iteration on \( n \):

\[
\begin{align*}
M_v \frac{\mathbf{v}^{n+1}_h - \mathbf{v}^n_h}{\Delta t} + R_\sigma \mathbf{\sigma}^{n+1/2}_h + B_v \frac{\mathbf{v}^{n+1}_h + \mathbf{v}^n_h}{2} &= 0 \\
M_\sigma \frac{\mathbf{\sigma}^{n+3/2}_h - \mathbf{\sigma}^{n+1/2}_h}{\Delta t} + R_v \mathbf{v}^{n+1}_h &= 0
\end{align*}
\]

✓ DG is very suitable for HPC context!
Quasi-explicit reformulation

\[
\begin{align*}
\mathbf{v}_h^{n+1} &= \mathbf{v}_h^n + \mathbf{M}_v^{-1}\Delta t \mathbf{R}_\sigma \mathbf{\sigma}_h^{n+1/2} \\
\mathbf{\sigma}_h^{n+3/2} &= \mathbf{\sigma}_h^{n+1/2} + \mathbf{M}_\mathbf{\sigma}^{-1}\Delta t \mathbf{R}_v \mathbf{v}_h^{n+1}
\end{align*}
\]

*UpdateVelocity*

*UpdateStress*

---

**Algorithm 1** : DIVA sequential

**Data** : \( N_h, \Delta t, N_t \)

**Result** : \( \mathbf{v}_h, \mathbf{\sigma}_h \)

\([\mathbf{v}_h^1, \mathbf{\sigma}_h^{3/2}] \leftarrow \text{Initialization}(N_h, \Delta t);\)

**for** \( n = 1 .. N_t \) **do**

**for** \( K = 1 .. N_h \) **do**

\( \mathbf{v}_{hK}^{n+1} \leftarrow \text{UpdateVelocity}(\mathbf{v}_{hK}^n, \mathbf{\sigma}_{hK}^{n+1/2}, \Delta t);\)

**end**

**for** \( K = 1 .. N_h \) **do**

\( \mathbf{\sigma}_{hK}^{n+3/2} \leftarrow \text{UpdateStress}(\mathbf{\sigma}_{hK}^{n+1/2}, \mathbf{v}_{hK}^{n+1}, \Delta t);\)

**end**

**end**
Algorithm 2: DIVA parallel (MPI)

Data: \( N_p, N_h, \Delta_t, N_t \)

Result: \( v_h, \sigma_h \)

\( N_{h_{\text{loc}}} \leftarrow \text{DomainDecomposition}(N_p) \);

\( [v_h^1, \sigma_h^{3/2}] \leftarrow \text{Initialization}(N_{h_{\text{loc}}}, \Delta_t) \);

for \( n = 1 \ldots N_t \) do

\( \sigma_{h_K}^{n+1/2} \leftarrow \text{Communication}_\text{MPI}(\sigma_{h_K}^{n+1/2}) \);

for \( K = 1 \ldots N_{h_{\text{loc}}} \) do

\( v_{h_K}^{n+1} \leftarrow \text{UpdateVelocity}(v_{h_K}^n, \sigma_{h_K}^{n+1/2}, \Delta_t) \);

end

\( v_{h_K}^{n+1} \leftarrow \text{Communication}_\text{MPI}(v_{h_K}^{n+1}) \);

for \( K = 1 \ldots N_{h_{\text{loc}}} \) do

\( \sigma_{h_K}^{n+3/2} \leftarrow \text{UpdateStress}(\sigma_{h_K}^{n+1/2}, v_{h_K}^{n+1}, \Delta_t) \);

end

end
Motivation: load imbalance issue

- **ORANGE** – UpdateVelocity kernels
- **RED** – UpdateStress kernels
- **GREY** – idle time (before communications)
Outline

1. MPI code implementation
2. Task-based programming
3. Numerical illustration
Task-based runtimes

Dense linear algebra libraries:

- shared memory: PLASMA [QUARK] / FLAME [SuperMatrix]
- distributed memory: DPLASMA [PaRSEC]
- accelerators: Chameleon [QUARK-StarPU] / DPLASMA [PaRSEC]

More irregular algorithms:

- Sparse direct and iterative methods – FP7: Exa2ct
- Fast algorithms (e.g. FMM, H-matrix) – H2020

- OpenMP, OpenACC with pragmas
- Taggre [TOTAL] for task aggregation
- OMPSs, X-KAAPI, CnC, . . .
Task-based programming

Principles

- Determine dependencies between tasks with the Bernstein’s conditions
- Specify the data modes: Read/Write

⇒ creation of a DAG (Direct Acyclic Graph)

Original control-flow becomes useless and can be an obstacle to efficiency

fun1(A: inout, B: out)
fun2(B: in, C: out)
fun3(A: inout, C: in)
Algorithme 3 : DIVA parallel (MPI)

Data : $N_p$, $N_h$, $\Delta t$, $N_t$

Result : $v_h$, $\sigma_h$

$N_{h_{loc}} \leftarrow \text{DomainDecomposition}(N_p)$;

$[v_h^{1/2}, \sigma_h^{3/2}] \leftarrow \text{Initialization}(N_{h_{loc}}, \Delta t)$;

for $n = 1..N_t$ do

$\frac{\sigma^{n+1/2}}{h_K} \leftarrow \text{Communication\_MPI}(\sigma_h^{n+1/2})$;

for $K = 1..N_{h_{loc}}$ do

$v_h^{n+1} \leftarrow \text{UpdateVelocity}(v_h^n, \sigma_h^{n+1/2}, \Delta t)$;

end

$\frac{v_h^{n+1}}{h_K} \leftarrow \text{Communication\_MPI}(v_h^{n+1})$;

for $K = 1..N_{h_{loc}}$ do

$\frac{\sigma^{n+3/2}}{h_K} \leftarrow \text{UpdateStress}(\sigma_h^{n+1/2}, v_h^{n+1}, \Delta t)$;

end

end

Data

- V: Velocity
- S: Stress
DIVA DAG

One iteration
Fine granularity

More than one domain per CPU
- exhibit deeper parallelism
- allow dynamic flexibility
- reduce the boundary size

Figure: Subdivision example
Outline

1. MPI code implementation
2. Task-based programming
3. Numerical illustration
Realistic test case:
- 3D elastic
- TTI (anisotropy)
- multi-layers

Hybrid discretization:
- unstructured tetrahedra
- \( \sim 800,000 \) cells
- P1-P2-P3 orders
⇒ both fine granularity and work-stealing are essential!
Figure: cache-coherent Non-Uniform Memory Access (ccNUMA) scheme
Numerical illustration

ccNUMA results - efficiency

![Graph showing ccNUMA results - efficiency](image-url)
Trace comparison

Figure: MPI-based $t = 2.517s$

Figure: PaRSEC version (NUMA-aware, granularity x6) $t = 2.060s$
Intel Xeon Phi architecture

(a) CPU

(b) MIC

Numerical illustration
Numerical illustration

Intel Xeon Phi results - efficiency

![Graph showing efficiency vs. number of threads for Perfect, PaRSEC, and MPI]

- Perfect
- PaRSEC
- MPI

Lionel BOILLOT (Inria)
Conclusion and perspectives

Context
- compute & exchange algorithm
- domain decomposition parallelism

Key-points
- fine granularity and work-stealing
- virtual process capability of PaRSEC

Results
Shared memory architectures (ccNUMA machine, Xeon Phi co-processor) ⇒ Next coming: distributed memory machines

Support by INRIA-TOTAL strategic action DIP (http://dip.inria.fr)