Alleviating performance bottlenecks on pre-exascale systems

G. Latu & GYSELA team
CEA, IRFM, France
collaborations: CEA/MdlS, Max Planck/Garching,
Bordeaux, Strasbourg Universities, INRIA
Outline

- **HPC trends**
  - Hardware & Programming issues

- **Gysela running on HPC**
  - Physics/numerics framework
  - Scaling on large machines
  - Intra-node optimization & issues

- **Preparing Gysela for exascale**
  - Task programming
  - Mini-apps
    - Gysela++ using OpenMP tasks
    - 1D-1V Vlasov-Poisson with tasks

- **Gysela targeting new features**
  - Edge/core interplay
  - Numerical challenges
HPC trends
Looking backward: terascale & petascale

- **Terascale** $10^{12}$ FLOPS, 9,000 cores
  - Mainly MPI applications, C/C++/Fortran

- **Petascale** $10^{15}$ FLOPS, 100,000 cores
  - Hybrid MPI+OpenMP popularizes
  - Performance modeling needed at large scale
HPC trends
Exascale is not ...

- **Just** a set of million cores
- **Minimal** programming support
- **Given** a LINPACK benchmark at one exaflops, all apps perform well → Work to do
- **Cheap** → X-billions dollars
- **Only** a dedicated power plant
- **Merely** a fight of GPUs against CPUs
HPC trends
Parallel algo. should stick to hardware

- Hardware evolve constantly

![Graph showing trends in FLOPs vs memory bandwidth on HPC servers, SC’16, John McCalpin]

- Gap between **data access** and **CPU speed** increased much
  - more FLOPS available per byte moved → **pressure** is on memory bandwidth
- One must change algorithms to cope with this **memory wall**:
  - reduce mem. moves
  - increase **computational intensity** (FLOPS per byte)
- Increasing number of **memory-bound** applications
  - applications tend to use a small fraction of the peak CPU [0.1% – 4%]
HPC trends
Timeline of Exascale: 2021-2023

- **US roadmap** (over 1 billion $ per year)
  - IBM/Nvidia/Mellanox
    - Power9 + **GPU** + NVRAM
  - Intel/Cray
    - based on Skylake’s successor ?

- **Chinese way** (over 1 billion $ per year)
  - 3 chinese manufacturers competing **homegrown processors**

- **EU roadmap** (over 1 billion $ total)
  - Which arch ?
    - ARM, classical chips or FPGA ?

- **Japan roadmap** (over 1 billion $ total)
  - Post-K machine: Fujistu
    - ARM-based

**First exascale machines:**

- ≈ 25-35MW consumption
- ≈ 30-60 **GFLOPS/watt**
  - factor 2x-10x still lacking

**many uncertainties ...**

hardware, prog. environment
HPC trends
Intra-node efficient algorithms & scalable codes

▶ Concurrency
  ▶ Exascale systems → billion of threads
  ▶ Design new multi-level parallel algorithms

▶ Latencies
  ▶ Count $10^2$ – $10^3$ cycles for mem access, $10^4$ for remote access
    → handling computation/comm. overlap, revisit main data structures
  ▶ Current algorithms have synchronization points (threads or MPI)
    → remove synchronizations to hide latency, e.g. use tasks, MPI-3

▶ Intra-node
  ▶ Re-evaluate existing methods (computational intensity ↗)
  ▶ Static load balancing is over: rely on runtime systems
  ▶ Reduce memory bandwidth pressure by improving locality
  ▶ Vectorizing, Instruction level parallelism
  ▶ Fine tuning of affinity and deployment

▶ Many other challenges
  ▶ IO, prog. models, faults, heterogeneity, mem. hierarchy, ...
Outline

▶ HPC trends
  ▶ Hardware & Programming issues

▶ Gysela running on HPC
  ▶ Physics/numerics framework
  ▶ Scaling on large machines
  ▶ Intra-node optimization & issues

▶ Preparing Gysela for exascale
  ▶ Task programming
  ▶ Mini-apps
    ● Gysela++ using OpenMP tasks
    ● 1D-1V Vlasov-Poisson with tasks

▶ Gysela targeting new features
  ▶ Edge/core interplay
  ▶ Numerical challenges
Investigation of core plasma performance by turbulence simulation

The GYSELA simulation tool (GYrokinetic SEmi-LAgragian code):

- Modelling of Tokamak plasma, targeting ITER (large device)
- Describing turbulence and transport turbulence limits plasma perf.
- Slightly collisional plasma
- Main equations: Vlasov, Poisson
- Heat & vorticity sources (mimics heating system)
- Collisional operators
- Kinetic electrons (instead of adiabatic e-)
- Accurate simulation $\to \times 10^6$ CPU hours
- Other EU codes: GENE, ORB5, ELMFIRE

ITER Tokamak under construction now
Tokamak kinetic model: 6D

- 3D in space: \((r, \theta, \varphi)\)
- 3D in velocity: \((v_\perp, \alpha, v_\parallel)\)

Time scale of particle gyration \(\alpha\) \ll time scale of turbulence

Gyrokinetic theory: 6D \(\rightarrow\) reduced to a 5D problem

Variables \((v_\perp, \alpha)\) replaced by adiabatic invariant \(\mu = \frac{mv_\perp^2}{2B}\)

Gyrokinetic model 5D+\(t\):

Distrib. function \(\bar{f}(r, \theta, \varphi, v_\parallel, \mu, t)\), Electric potential \(\phi(r, \theta, \varphi, t)\)

Vlasov/Boltzmann: \[\frac{\partial \bar{f}}{\partial t} + \frac{d x^2}{d t} \nabla_x \bar{f} + \frac{d v_\parallel}{d t} \frac{\partial \bar{f}}{\partial v_\parallel} = \text{(collisions)} + \text{(sources)}\]

Poisson: \[-\frac{1}{n_0(r)} \nabla_\perp \left[ \frac{n_0(r)}{B_0} \nabla_\perp \phi(r, \theta, \varphi) \right] + \frac{1}{T_e(r)} \left[ \phi(r, \theta, \varphi) - \langle \phi \rangle(r) \right] = \]

\[\frac{1}{n_0(r)} \int \int \mathcal{J}_v J_0(k_\perp \sqrt{2\mu}) (\bar{f} - \bar{f}_{\text{ref}})(r, \theta, \varphi, v_\parallel, \mu) \, dv_\parallel \, d\mu.\]
Numerical scheme
Strang splitting to solve Vlasov

- **Simplified** view of gyrokinetic Vlasov equation (costly part):

\[
\frac{\partial \tilde{f}}{\partial t} + \frac{d\tilde{x}}{dt} \nabla_{\tilde{x}} \tilde{f} + \frac{dv_{\parallel}}{dt} \frac{\partial \tilde{f}}{\partial v_{\parallel}} = 0
\]

(collisionless)

(no source)

- **Strang splitting** (decomposed into directional advections), \(\mu\) acts as a parameter

\[
\partial_t \tilde{f} + \dot{v}_{\parallel} \partial_{v_{\parallel}} \tilde{f} = 0 \quad (\hat{v}_{\parallel} \text{ operator, } \frac{\Delta t}{2})
\]

\[
\partial_t \tilde{f} + v_{\parallel} \partial_{\phi} \tilde{f} = 0 \quad (\hat{\phi} \text{ operator, } \frac{\Delta t}{2})
\]

\[
\partial_t \tilde{f} + v_{GC} \cdot \nabla_{\perp} \tilde{f} = 0 \quad (\hat{r} \theta \text{ operator, } \Delta t)
\]

\[
\partial_t \tilde{f} + v_{\parallel} \partial_{\phi} \tilde{f} = 0 \quad (\hat{\phi} \text{ operator, } \frac{\Delta t}{2})
\]

\[
\partial_t \tilde{f} + \dot{v}_{\parallel} \partial_{v_{\parallel}} \tilde{f} = 0 \quad (\hat{v}_{\parallel} \text{ operator, } \frac{\Delta t}{2})
\]
The semi-Lagrangian method backward version

1) \( f \) conserved along characteristics

2) Find the origin \( X^* \) of the characteristics ending at one grid pt

3) Interpolate value at origin \( X^* \) from \( \rightarrow \) Interpolation method needed

Typical interpolation scheme:
- Cubic spline

No CFL restriction on \( \Delta t \)
Main data $\tilde{f}$: 5D array $(r, \theta, \varphi)$ space, $(v_\parallel, \mu)$ velocity

One simplified time step (3 domain decomps.):

- **MPI process**
  - $\mu_0$
  - Transpose
  - $\mu_1$
  - Transpose
  - $\mu_N$
  - Transpose

- **Transverse**
  - $\nu_\parallel$ advection
  - $\varphi$ advection
  - $r, \theta$ advection

- **Gyroaverage**
  - Poisson
  - Gyroaverage

---

[Latu & al., PASC, 2016]
Parallel algorithm  
1D advection $\varphi$ direction

Input: $\tilde{f}^*(r, \theta, \varphi, v_\parallel, \mu), \Phi(r, \theta, \varphi)$  
Output: $\tilde{f}^\varphi(r, \theta, \varphi, v_\parallel, \mu)$

for $\mu$ do in parallel MPI /* One MPI communicator per $\mu$-value */  
for $r$ do in parallel MPI  
  for $\theta$ do in parallel MPI
    for $\theta$ do in parallel OpenMP
      for $v_\parallel$ do
        $\Delta \varphi \leftarrow$ displacement along $\varphi$ direction (depends on $\Phi$)
        for $\varphi$ do  
          $\tilde{f}^\varphi(r, \theta, \varphi, v_\parallel, \mu) = \text{interpolate}(\tilde{f}^*(r, \theta, \varphi - \Delta \varphi, v_\parallel, \mu))$

[Latu & al., PVM/MPI, 2007]
GYSELA competitive in terms of physics targeting finer approximations

**Plasma physics** topics:

- Interaction fast particles & turbulence
- Characterisation of turbulent transport
- Comparison with experiments

Plot: Cross-section of the electric potential $\phi(r, \theta, \varphi = 0)$

Everlasting challenge: larger domain size

Plot: 3D view of the electric potential $\phi(r, \theta, \varphi)$

Current aims: solve numerical issues

[Grandgirard & al., CPC, 2016]
Performance on supercomputers [2013]

- Good **strong** scaling \(\approx 60\%\) efficiency at 65k cores (Curie & Turing)
- Good **weak** scaling \(\approx 91\%\) efficiency at 459k cores (Juqueen machine)

Load imbalance appears at large scale

- need to balance the load across different subroutines + improve // algos

Communication costs originating from:

- heavy data transposition (distrib function) + collective communications
New constraint: **larger number** of threads (threads per node $\geq 48$)

Expecting improved **intra-node load balancing** $\rightarrow$ exascale

- Work done (2016): remove synchronizations, redesign // algorithms, fuse OpenMP regions, use **profiling tool**, hyper-threading

- Result: work done: $-38\%$ exec. time (Haswell)

---

### After Optimisation

<table>
<thead>
<tr>
<th>After Optimisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Adv. vpar</td>
</tr>
<tr>
<td>Transpose</td>
</tr>
<tr>
<td>Diffusion</td>
</tr>
<tr>
<td>2D Advection</td>
</tr>
<tr>
<td>Transpose</td>
</tr>
<tr>
<td>1D Adv. phi</td>
</tr>
<tr>
<td>Field solver</td>
</tr>
<tr>
<td>Deriv. Comp.</td>
</tr>
</tbody>
</table>

---

Paraver plots

[https://tools.bsc.es/paraver](https://tools.bsc.es/paraver)

[Latu & al., PASC, 2016]
Intra-node optimization (2/4)
vectorizing GYSELA (2016-2017)

- **Targets:** INTEL KNL & Skylake:
  Vector register width = 512-bits (8 double precision float numbers)

- **Profiling tools** required: INTEL Vtune, INTEL Advisor

- Optimizing GYSELA: code refactoring → help the compiler vectorizing
  - inlining, conditionals/ifs, small vectors as local variables
  - SIMD directives, loop splitting

- Change several algorithms in GYSELA
  - Avoid long-strided accesses in **memory**, promote unit-strided → copy small buffers forward and backward
  - Grouping a set of LU and tridiagonal solve
    explicit use of multiple RHS and LHS (**vectorizing**)

- **Deployment** MPI/OpenMP (NUMA effects)
  - 4 MPI processes of 34 threads within a node (68 cores)
Trend: large *computational intensity* is advantageous → expected gain of *high-order* methods tends to increase

Interpolation scheme: **High-order** Lagrange instead of cubic *splines*

- Lagrange polynomials degree 5 & 7 selected (accuracy)
- Lagrange involves **extra costs**, avg cost per point:

<table>
<thead>
<tr>
<th>Kind of interpolation</th>
<th>Multiply</th>
<th>Add</th>
<th>Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D spline</td>
<td>26</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>1D Lagrange 5(^{th})</td>
<td>30</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>1D Lagrange 7(^{th})</td>
<td>48</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>2D spline</td>
<td>60</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>2D Lagrange 5(^{th})</td>
<td>90</td>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td>2D Lagrange 7(^{th})</td>
<td>144</td>
<td>122</td>
<td>0</td>
</tr>
</tbody>
</table>

However

- Vectorization is easier for Lagrange formula → compiler works better → Lagrange 5\(^{th}\) faster than splines

Lagrange included in **GYSELA**
Intra-node optimization (4/4)
Vectorizing + High-order (2016-2018)

- Improvement due to **vectorizing**, **improved memory accesses**, interpolate with **Lagrange** polynomial (degree 7th),
  - Benchmarks: INTEL Broadwell and KNL and Skylake
    - Speedup of **7x** on Vlasov solver exec. time 🙄
    - Speedup of **1.8x** up to **3x** gained on total exec. time 😊

- **Intra-node** optimization is a main issue for exascale
  - [Latu & al., SBAC-PAD’18 proc.]

- **But** at large scale, bottlenecks remain:
  - **comm. costs**, **load imbalance**
    - Vlasov has > 45% of **communication costs** at large scale [2017]
    - Communication/compute **overlap** required
Up to 32k cores (512 KNL nodes). Irene machine, CCRT, Bruyères, France.
Domain size $256 \times 512 \times 128 \times 64 \times 64$ - Medium case

Good behavior of large scale:
63% of relative efficiency at 33k cores [below numbers of 2013]
but #cores increases!
Outline

▶ HPC trends
  ▶ Hardware & Programming issues

▶ Gysela running on HPC
  ▶ Physics/numerics framework
  ▶ Scaling on large machines
  ▶ Intra-node optimization & issues

▶ Preparing Gysela for exascale
  ▶ Task programming
  ▶ Mini-apps
    ● Gysela++ using OpenMP tasks
    ● 1D-1V Vlasov-Poisson with tasks

▶ Gysela targeting new features
  ▶ Edge/core interplay
  ▶ Numerical challenges
Task programming: Principles

▶ Pb: **intra-node load balancing** nb of computing units \(\uparrow\) (60-2000)
▶ Pb: **synchronization** costs MPI and OpenMP
▶ Solution: **task** → harmless synchronizations + eases overlap of communication
do k=1, min(mb,nb)
   !$omp task depend(inout:a(k,k))
   call dgeqrt(a(k,k))
   !$omp end task
   do j=k+1,nb
      !$omp task depend(in:a(k,k)) depend(inout:a(k,j))
      call dgemqrt(a(k,k), a(k,j))
      !$omp end task
   end do
   do i=k+1, mb
      !$omp task depend(inout:a(k,k), a(i,k))
      call dtsqrt(a(k,k), a(i,k))
      !$omp end task
      do j=k+1, nb
         !$omp task depend(in:a(k,k),a(i,k))
         !$omp depend(inout:a(k,j),a(i,j))
         call dtmqrt(a(k,k), a(i,k), a(k,j), a(i,j))
         !$omp end task
      end do
   end do
end do
Adapt complex HPC codes to Exascale
Mini-apps

- Complex math. schemes, new HPC techno
  - Require large manpower, overhauling old HPC designs
  - Design choices may be reconsidered along the way

- Mini-applications
  - Allows one to focus to **specific** issues
  - Flexible tools that enable short learning cycles at an affordable cost
  - Short enough to be developed in few weeks
  - Helps designing the HPC application → combined challenges

- Expectations for Gysela in the future
  - New physics, new geometry
  - Non-uniform grid in phase space
    → Scrape-off layer with extreme gradient
  - Fit well on HPC systems
  - Several **mini-apps** under construction
Mini-app-1: Gysela++

Scope

1) Numerical scheme
   - Non-uniform grids in poloidal plane
   - Model: Drift-kinetic 4D (single $\mu$ value)

2) Parallelization strategy
   - Parallelize along all 4 dimensions
   - Dedicate one thread to communications
   - Ghost areas to manage the interpolations
     $\rightarrow$ avoid transpose comm.

3) Evaluating OpenMP 4.0 - task features
   - OpenMP tasks balancing the load within a node
   - MPI for parallelism between nodes (overlapped)
   - C++ language: specialization of routine depending on localisation
   - Each task is associated to a 4D tile data structure $\rightarrow$ locality

[N. Bouzat PhD]

G. Latu & al.
Perf. bottlenecks on pre-exascale systems  11/12/2018
Mini-app-1: Gysela++
Reduced grid and tiles

A reduced poloidal grid

A partition with 9 tiles
Mini-app-1: Gysela++
Machinery within each MPI process

- Comm. thread $T_0$, Controller $T_1$, Workers $T_2...n$
- MPI_THREAD_FUNNELED mode $\rightarrow$ efficient

Communications:
- 3D data redistrib
- Halo exchange
Mini-app-1: Gysela++
Gantt Chart

Trace of one iteration on one MPI process

- Task traces plotted with ViTE tool
  http://vite.gforge.inria.fr/
Pros

- Overlapping is improved
- **Reduced grid** and tiles are working properly
- 4D domain decomposition provides scalability

Cons

- Computation kernels not optimized yet
- Tiles+Ghosts induce memory overheads
- The controller thread ($T_1$) involves **prog. difficulties**
  next mini-app avoids the controller

**Weak scaling Gysela++**

- 256x256x64x64 (1 process / 12 cores)
- 1024x1024x256x512 (512 processes / 6144 cores)
Mini-app-2: Vlasov-Poisson 1D
Scope

1) Setup
- Model: Vlasov-Poisson (1D \( x \) - 1D \( v \))
- Uniform grid along \( x \) and \( v \)
- Semi-Lagrangian scheme, 2D interpolations

2) Evaluating OpenMP 4.5 - task features
- Using KOMP runtime (close to INTEL runtime)
  because interesting features implemented
  - OpenMP tasks with priorities to change the schedule
  - Allow trace generation for visualization purposes
  - C language fully supported, Fortran may lack some functionalities

3) Parallelization strategy
- Each task is associated to a 2D tile data structure → locality
- Parallelize (MPI+OpenMP) along tile dimension
- Each tile has dimension \( L_x \times L_v \) : optimization parameter
- Ghost areas to manage the interpolations (\( \Delta t \) small → CFL)
- Ghost exchanges are sped up: treat separately inner/border tiles

[J. Richard Postdoc]
Input: $f^n$
Output: $f^{n+1}$

MPI Reduce $f^n \rightarrow \rho^n$ density, $J^n$ current
Field solver: compute $E^{n+1/2}$ (Poisson-Ampere eqs)
MPI Broadcast $E^{n+1/2}$

Launch Send/Receive for ghost zones $f^n$
In each tile, 2D advections for interior points $f^n \rightarrow f^{n+1}$
Wait receive for ghost zones in each local tile $f^n$
In each tile, 2D advections for border points $f^n \rightarrow f^{n+1}$
Wait send for ghost zones $f^n$ in each local tile
In each tile, Local integrals along velocity to get $\rho^{n+1}$
In each tile, Swap buffers for $f$ for time steps $n$ and $n+1$
Trace of one iteration on one MPI process among 64 processes (KNL testbed)

- MPI stuff done on thread 0
- Task scheduler induces overheads (→ task grain)
Trace of one iteration on one MPI process among 128 processes (KNL testbed)

- MPI stuff done on one thread at a time (MPI_THREAD_SERIALIZED)
- **Performance gains**, thanks to
  - visualization + priority + task split/merge + shorten critical path
Non-blocking comm. is of utmost importance to scale.
Delegate MPI comm. to non-master thread is shortening critical path.
Task version proves better performance than for-loop version.
Tasks give a good practical solution if Task priority is handled.
Visualization is available for perf. analysis.
Task size can be parameterized (to lower tasks overhead).
Benefits of tasks would be even higher for a non-uniform task granularity for operators executing concurrently.
Outline

▶ HPC trends
  ▶ Hardware & Programming issues

▶ Gysela running on HPC
  ▶ Physics/numerics framework
  ▶ Scaling on large machines
  ▶ Intra-node optimization & issues

▶ Preparing Gysela for exascale
  ▶ Task programming
  ▶ Mini-apps
    ● Gysela++ using OpenMP tasks
    ● 1D-1V Vlasov-Poisson with tasks

▶ Gysela targeting new features
  ▶ Edge/core interplay
  ▶ Numerical challenges
Plasma edge → critical sources of sheared flows

**Physics issue**

- **Collisional damping:**
  Neoclassical flow → $v_\theta \sim \nabla T/eB$

- **Core** (confined) plasma:
  Radial force balance → $E_r \sim \nabla p/en$

- **Unconfined region** ("Scrape-Off Layer"):
  // plasma-wall interaction → $E_r \sim -\nabla T/e$

- **Ion orbit losses** → plasma polarization $E_r$

- **Turbulence self-generated flows**
  Large fluctuations at the edge → large $E_r$?

- **Source of neutrals** → collisional friction

**DONE**

**Collision operator**

**Core plasma**

**TODO now**

**SOL plasma**

**Boundary conditions**

**MUCH better**

**Ion & electron turbulence**

**Neutrals & impurities**

+ access to ITER relevant parameters ($\rho_i/a \sim 2.10^{-3}$)

**Courtesy of Y. Sarazin**
Relevant radial boundary conditions

Radial domain extended to $r/a>1$ in GYSELA

- **Simplified** original unconfined (SOL) region
  - Limiter $\rightarrow$ immersed boundary (penalization technique)
  - Prescribed divertor-plasma interaction in $//\text{ direction}$

![Diagram of radial boundary conditions](image)

[Caschera PhD (2018), Dif-Pradalier (2018)]

![Graph of RMS of $\delta n/n$ vs Normalised radius $\rho$](image)
Numerical challenges – new mesh/code

- Arbitrary & consistent magnetic equilibrium
  - Non-circular plasma
  - X-point singularity
  - Strong anisotropy, open field lines

- From the plasma core to the Scrape-Off Layer
  - Cope with large variation of temperature → 5D patches?
    - Temperature varies by orders of magnitude from core to edge

Cf. e.g. [Jarema CPC (2017)]
Two main approaches considered

(1) Mapped grids

- Patch 1
- Patch 2
- Patch 3
- First wall
- Last closed magnetic surface
- Immersed Boundary Conditions

(2) Cartesian grid

- First wall
- Last closed magnetic surface

- Revisit semi-Lagrangian method:
  - (1) mapped logical domain, non-uniform grids
  - or (2) cartesian grid \((R,Z)\), non-uniform grids

- Adapted Poisson solver: **boundary** conditions

- **Scalable** algorithms \(\rightarrow\) versatile domain decomposition
Conclusion: How to address pre-Exascale and Exascale systems?

- **Modify slightly** existing applications
  - More scalable algorithms, overlap some communication schemes
  - Increase computational intensity, increase memory locality
  - Auto-tuning, vectorizing
  - Incorporate in kernels: new programming model, offload to GPU
    *Not sufficient* to address exascale, but right direction

- **Overhaul a few** applications for exascale systems
  - Gamble: high risk / high payoff / large **costs**
  - Interdisciplinary efforts:
    Physics, Mathematics, Computer science
  - Re-design: main numerics/algorithms targeting one architecture (or two)
  - GyselaX is expecting to follow this path
Conclusion: Aiming at a new gyrokinetic code

- **Developp/improve** numerical methods and mini-apps
  - Math: multi-resolution mesh, 4D advections, Poisson, geometry, boundary cond, fast e-, large gradients
  - HPC: tasks, 5D domain decomposition, C++, overlap, vectorization, in-transit data analysis

- **Write** GyselaX
  - Start in 2019
  - Gamble: high risk / high payoff / large **costs**
  - Interdisciplinary efforts: Physics, Mathematics, Computer science