Outline

• Motivation
• Periscope overview
  – Concepts and features
• GENE analysis results
  – Applying Periscope
  – Single Core Performance analysis results
  – MPI performance analysis results
• Conclusion
Motivation

• Common performance analysis procedure on Power6 systems
  – Use Tprof to pinpoint time-consuming subroutines
  – Use Xprofiler to understand call graph; mpitrace for MPI comm
  – Use hpmcount (libhpm) to measure HW Counters

• Issues
  – Routine, error-prone and time-consuming
  – Requires deep HW knowledge
  – Mostly post-development process
  – Learning multiple tools required
  – Hard to map bottlenecks to their source code location

• Solution
  – Automate the performance analysis
  – Integrate parallel application development and performance analysis within the same IDE
Periscope

- **ISAR Project**
  - Extending functionality of Periscope, porting to new architectures, productization
  - BMBF funded, IBM as a partner

- **Iterative online analysis**
  - Configure, measure and evaluate raw performance data on the fly
  - No raw performance data stored, no trace files

- **Distributed architecture**
  - Scale together with application using multiple distributed hierarchical agents
  - Reduce performance data online

- **Automatic bottlenecks search**
  - Do analysis automatically – search for most important bottlenecks
  - Based on performance optimization experts' knowledge

- **Enhanced Eclipse-based GUI**
  - Bring discovered bottlenecks back to the programmers environment
  - Map results to the code lines

- **Instrumentation**
  - Fortran, C/C++; MPI / OpenMP / Hybrid, Automatic Overhead Control
Periscope distributed architecture

- Graphical User Interface
  - Eclipse-based GUI
- Interactive frontend
  - Analysis control
  - Agents network
- Monitoring Request Interface
- Application
Automatic search for bottlenecks

- Analysis automation through tuning knowledge formalization
  - Potential performance bottlenecks $\rightarrow$ properties
  - Efficient search algorithms $\rightarrow$ search strategies

- Performance property
  - Tries to map execution domain inefficiencies to algorithm domain
  - Condition
  - Confidence
  - Severity

- Performance analysis strategies
  - Itanium2 Stall Cycle Analysis
  - IBM POWER6 Single Core Performance Analysis
  - MPI Communication Pattern Analysis
  - Generic Memory Strategy
  - OpenMP-based Performance Analysis
  - Scalability Analysis – OpenMP codes
GENE overview

- **What is GENE?**
  - “Plasma microturbulence code which can be used to compute gyroradius-scale fluctuations and the resulting transport coefficients in magnetized fusion/astrophysical plasmas” *
  - Solves the nonlinear gyrokinetic equations on a fixed grid in five-dimensional phase space (plus time)

- **Statistics and features**
  - Hybrid MPI/OpenMP parallelization approach
  - Parallelized over all space coordinates
  - Highly efficient: adapts to given hardware and problem size, chooses timestep in an optimal way
  - 129 source files, 49722 lines of code (FORTRAN 90)
  - Currently 11 developers; collaborators and users from over 24 institutions

- **Runs on a large number of computer architectures**
  - IBM BlueGene/P, IBM Power5/6, SGI Altix 4700, Cray XT-XE6, various Linux clusters

* [www.ipp.mpg.de/~fsj/gene/](http://www.ipp.mpg.de/~fsj/gene/)
Preparing GENE for analysis with Periscope

- **Instrument**
  
  bin/vip_p6.mk:

  ```
  ...
  ###################################################################
  ### COMPILER & COMPILER FLAGS ###
  ###################################################################
  FC = psc_instrument -s gene.sir mpxlf95_r -l/usr/lpp/ppe.poe/include/thread64
  CC = psc_instrument -s gene.sir mpcc_r -l/usr/lpp/ppe.poe/include/thread64
  ...
  ```

  %make

- **Run**
  
  %psc_regrsv
  %psc_frontend --apprun=../../gene_vip_p6 --mpinumprocs=$NP --debug=6 --maxfan=8 --sir=../../appl.sir
  --strategy=P6BF_Memory --propfile=properties_${NP}.psc

- **Visualize**
  
  %eclipse
GENE single core performance analysis with Periscope...

Very good performance 😊
But there is always room for improvement...

All properties above 10% threshold

<table>
<thead>
<tr>
<th>Name</th>
<th>Filename</th>
<th>RFL</th>
<th>Severity</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>97</td>
<td>100.00</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>133</td>
<td>99.98</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>77</td>
<td>99.98</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>133</td>
<td>82.45</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>77</td>
<td>82.45</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>133</td>
<td>82.45</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>179</td>
<td>75.19</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>274</td>
<td>70.07</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>287</td>
<td>66.90</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>Hot spot of the application - very good performance</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>179</td>
<td>62.03</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Floating point instructions execution slowed down due to register conflicts</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>287</td>
<td>53.36</td>
<td>4, 5, 6, 7</td>
</tr>
<tr>
<td>Floating point instructions execution slowed down due to register conflicts</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>274</td>
<td>49.25</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Floating point instructions execution slowed down due to register conflicts</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>287</td>
<td>46.68</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Floating point instructions execution slowed down due to register conflicts</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>133</td>
<td>46.68</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Memory access pattern : Insufficient cached data reuse with 64.1714%</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>97</td>
<td>19.53</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Memory access pattern : Insufficient cached data reuse with 64.0907%</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>133</td>
<td>19.52</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Memory access pattern : Insufficient cached data reuse with 64.0907%</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>77</td>
<td>19.52</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Memory access pattern : Insufficient cached data reuse with 65.4889%</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>133</td>
<td>15.29</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Memory access pattern : Insufficient cached data reuse with 66.9099%</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>274</td>
<td>11.93</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Cycles lost due to floating point queue is full</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>97</td>
<td>10.87</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Cycles lost due to floating point queue is full</td>
<td>/tmp/pyooGENE-1.5/src/time_scheme.F90</td>
<td>133</td>
<td>10.87</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Cycles lost due to floating point queue is full</td>
<td>/tmp/pyooGENE-1.5/src/calc_rhs.F90</td>
<td>274</td>
<td>10.87</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>Cycles lost due to floating point queue is full</td>
<td>/tmp/pyooGENE-1.5/src/value_comp.F90</td>
<td>287</td>
<td>10.04</td>
<td>0, 1, 2</td>
</tr>
</tbody>
</table>
But there is always room for improvement…
Region wasting most of the time

Loop region from GENE: calc_rhs.F90:274:

!computation is done in sub-blocks of g-like arrays to optimize cache usage
do iblock=1,nblocks
  lbg1 = (iblock-1)*lbg0+1
  lbg2 = lbg1+lbg0-1
  if(.not.with_g_update) rhs_block=>a_rhs(:,:,iblock)
  ....
  enddo

Periscope:
Memory access pattern: Insufficient cached data reuse with only 66% reuse factor

- bandwidth 17985.1 MB/s
- Avg cycles per L1 miss 34
- Cycles lost due to cache 11% of the phase execution (15% of the region time)
- L1 miss rate 1.2%
- L2 miss rate 0.7%
- L3 miss rate 39%
- L1 prefetch ratio 33%
- Loaded+stored bytes 15201 MB
subroutine nextstage_g(timestep, k_plus, g_old, g_new)
  real, intent(in) :: timestep
  complex, dimension(li1:li2, lj1:lj2, lk1:lk2, ll1:ll2, lm1:lm2, ln1:ln2), intent(in) :: k_plus, g_old
  complex, dimension(li1:li2, lj1:lj2, lk1:lk2, ll1:ll2, lm1:lm2, ln1:ln2), intent(inout) :: g_new
  PERFON('rkupd')
  g_new = g_old + timestep*k_plus
  PERFOFF
end subroutine nextstage_g

Periscope:
Memory access pattern: Store queue flooded, too many consequent stores
Streaming from memory (4169.13 MB/s); Severity 2.55%

* bandwidth        4200 MB/s
* Avg cycles per L1 miss  575
* Cycles lost due to cache 2.55% of the phase execution (60% of the region time)
* L1 miss rate    1.58%
* L2 miss rate    10%
* L3 miss rate    100%
* L1 prefetch ratio 100%
* Loaded+stored bytes 151 MB
* Store queue full rejects per store reference 70%
GENE MPI performance analysis with Periscope...

No properties found 😊
If it can’t be better, make it worth...

Inefficient parallelization choice

<table>
<thead>
<tr>
<th>Name</th>
<th>Filename</th>
<th>RFL</th>
<th>Severity</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>6.89</td>
<td>7, 12</td>
</tr>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>10.48</td>
<td>1</td>
</tr>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>5.10</td>
<td>11, 13</td>
</tr>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>10.02</td>
<td>14, 15</td>
</tr>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>14.02</td>
<td>0, 2, 3, 6, 14, 15</td>
</tr>
<tr>
<td>Excessive MPI time due to late process in allreduce</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>5.37</td>
<td>11, 13</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>523</td>
<td>10.77</td>
<td>1</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>15.87</td>
<td>3, 13</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>32.70</td>
<td>11</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>14.02</td>
<td>14, 15</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>14.60</td>
<td>14</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>14.60</td>
<td>14</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>8.79</td>
<td>0, 2, 3, 6, 14, 15</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>7.19</td>
<td>7, 12</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>21.15</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>274</td>
<td>22.39</td>
<td>9, 12</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>453</td>
<td>8.66</td>
<td>0, 1, 4, 5, 8, 9</td>
</tr>
<tr>
<td>Excessive MPI communication time</td>
<td>/u/dng/GENE-1.5/src/comm.F90</td>
<td>453</td>
<td>11.67</td>
<td>2, 3, 6, 7, 12, 13</td>
</tr>
</tbody>
</table>
Thank you for your attention!

- **GENE** – very good performance!
  - But there is always room for improvement

- **Periscope**
  - Highly scalable solution for automated performance analysis
  - Delivers performance problems not the performance data
  - Tries to formulate problems in programmers language
  - Covers major performance aspects:
    - MPI synchronization problems, time spent
    - Single core performance
    - OpenMP overheads, scalability
  - Brings performance analysis results directly into IDE
  - Available on POWER6, BlueGene/P (only MPI), SGI Altix 4700, x86-based architectures (only MPI)

- **Further information:**
  - Periscope web page: http://www.lrr.in.tum.de/periscope