Towards an Energy-Aware Scientific I/O Interface

Stretching the ADIOS Interface to Foster Performance Analysis and Energy Awareness

Julian M. Kunkel, Timo Minartz, Michael Kuhn, Thomas Ludwig

julian.martin.kunkel@informatik.uni-hamburg.de

Scientific Computing
Department of Informatics
University of Hamburg

08.09.2011

informatik die zukunft

informatik die zukunft

Motivation

Goal: Conserving of energy

- Hardware components can be put into a low power state.
  - But transitions between power states are time consuming.
- Intelligent switching of states is important.
  - Avoid (minimize) slow down of programms.
  - Induced noise endangers synchronization of processes.

Intelligent switching of states

- Knowledge of future program activity is required.
- Automatic vs. manual switching.
  - The system has limited information about future activity.
  - Developers have an idea about program behavior.

Motivation

Problem of manual annotation

- Tedious work
  - Developer must think about future activity.
- Error-prone
  - Sometimes annotations are not correct.
- Benefit for the user?
  - Convince developers to use the “new” interface.

Proposed solution

Extend an existing I/O interface to support **annotated phases**.
A library **analyzes** phases at runtime and **controls** hardware.

Extension of an existing I/O interface

Benefit of the extended interface/library

- Improve knowledge to estimate I/O time – optimize behavior.
  - Caching and background optimizations get more time.
- Available phase information can be given to performance tools.
  - Performance analysis is enriched with phase information.
- Automatic control of power states in the devices.
  - Reduce energy consumption.

Adaption of the interface

Threefold benefit of the light-weight interface might convince users.

Introduction of ADIOS

Adaptable IO System

- Alternative high-level I/O interface.
  - Annotations of variables similar to HDF5.
- Offers various back-ends: POSIX, MPI-IO, NULL or in-situ vis.
- BP file format.
  - Throughput oriented, avoids synchronization.
  - An ADIOS file may be represented by one or multiple objects.
  - Easy conversion of BP files into NetCDF or HDF5.
- XML specification of variables and run-time parameters.
  - Adapt programs to the site's file system without code adjustment.
  - Translate XML into C or Fortran code to read/write data.

Efficient I/O

Caching

- ADIOS aggressively caches data.
- Write-behind during computing phases.
- Function call indicates the speed of iterative programs.

User control in the XML

- Pick the best suitable backend for a supercomputer and task.
- Define cache size.
- Request to store derived data (histograms).

Listing 1: Sketched ADIOS code

```
1     int NX = 10, NY = 10, NZ = 100;  double matrix[NX][NY][NZ];
2     MPI_Comm comm = MPI_COMM_WORLD; int64_t adios_handle;
3     int adios_err; uint64_t adios_groupsize, adios_totalsize;
4
5     MPI_Init(&argc, &argv); MPI_Comm_rank(comm, &rank);
6     adios_init("example.xml");
7
8     for (t = 0; t < 10 ; t++) {
9       adios_start_calculation();
10      /* computation */
11      adios_stop_calculation();
12      /* MPI communication */
13      adios_open(&adios_handle, "fullData", "testfile.bp", t == 0
            ↪ ? "w": "a", &comm);
14 #include "gwrite_fullData.ch"
15      adios_close(adios_handle);
16      /* indicate progress for write-behind */
17      adios_end_iteration();
18    }
19
20    adios_finalize(rank); MPI_Finalize(); return 0;
```

Listing 2: ADIOS example code – `gwrite_fullData.ch`

```
1 adios_groupsize = 4 \
2                   + 4 \
3                   + 4 \
4                   + 8 * (NX) * (NY) * (NZ);
5 adios_group_size (adios_handle, adios_groupsize, &adios_totalsize);
6 adios_write (adios_handle, "NX", &NX);
7 adios_write (adios_handle, "NY", &NY);
8 adios_write (adios_handle, "NZ", &NZ);
9 adios_write (adios_handle, "matrix_data", matrix);
```

This code is automatically generated from the XML.

ADIOS XML code

```
<adios-config host-language="C">
  <adios-group name="fullData" coordination-communicator="comm"
    time-index="iteration">
    <attribute name="description" path="/fullData"
      value="Global array of memory data" type="string"/>
    <var name="NX" type="integer"/>
    <var name="NY" type="integer"/>
    <var name="NZ" type="integer"/>
    <var name="matrix_data" gwrite="matrix"  type="double"
      dimensions="iteration,NX,NY,NZ"/>
  </adios-group>

  <analysis adios-group="fullData" var="matrix_data"
    min="0" max="3000000" count="30"/>
  <method group="fullData"    method="MPI"/>
  <buffer size-MB="80" allocate-time="now"/>
</adios-config>
```

CIAO interface

Extension to ADIOS

- CIAO is used to refer to the modified functions.
- Classification into calculation, communication and I/O phases.
- Phases are named and span a longer period of execution.
  - Names encode high-level semantics.
  - The same name can be used to encode similar behavior.

Additional features

- Analyze phase invocation to understand the workflow.
- Characterize every named phase:
  - Time, energy, performance (CPU, network utilization).
  - Possibly this enables to classify the phases automatically!
- Trigger power state and I/O behavior if the change is promising.

Listing 3: CIAO example code

```
1  adios_init("example.xml");
2
3  ciao_open(...);
4  /* read input */
5  ciao_close(...);
6
7  ciao_start_calculation("pre-processing");
8  /* pre-process input */
9  ciao_end_calculation();
10
11 for (t = 0; t < 10 ; t++) {
12   ciao_start_calculation("iteration");
13   /* computation */
14   ciao_end_calculation();
15
16   ciao_start_communication("exchange-neighbour");
17   /* communication */
18   ciao_end_communication();
19
20   ciao_open(&adios_handle, "fullData", "testfile.bp", t == 0 ?
         ↪ "w": "a", &comm);
21 #include "gwrite_fullData.ch"
22   ciao_close(adios_handle);
23 }
24 adios_finalize(rank);
```

Characterization of phases

Prediction of phase characteristics

- Characteristics of repeated invocation might be similar:
  - Historic knowledge across program runs
  - The last (or average) characteristics
  - Minimum values (to avoid overestimation)
- The user can offer hints in the XML to set the predictor.

Estimation of program workflow

- Sequence of phase transitions could be tracked in CIAO.
- Predict future phases to estimate future utilization.

```
<adios-config host-language="C">
  ...
  <buffer size-MB="80" allocate-time="now"/>
  ...
  <estimation debug="statistics">
    <inter-phase method="STOCHASTIC" accept-threshold="95%">
    <phase name="iteration"       method="MIN"/>
    <phase name="post-processing" method="HISTORIC"/>
  <estimation/>
</adios-config>
```

Benefit for analysis tools

Phase knowledge enriches profiling and tracing

- Aggregate profile for each phase individually.
- Restrict analysis to phases of interest.

State of the art

- Phases are already known in performance analysis (TAU, ...)
- But they are just used for that purpose.

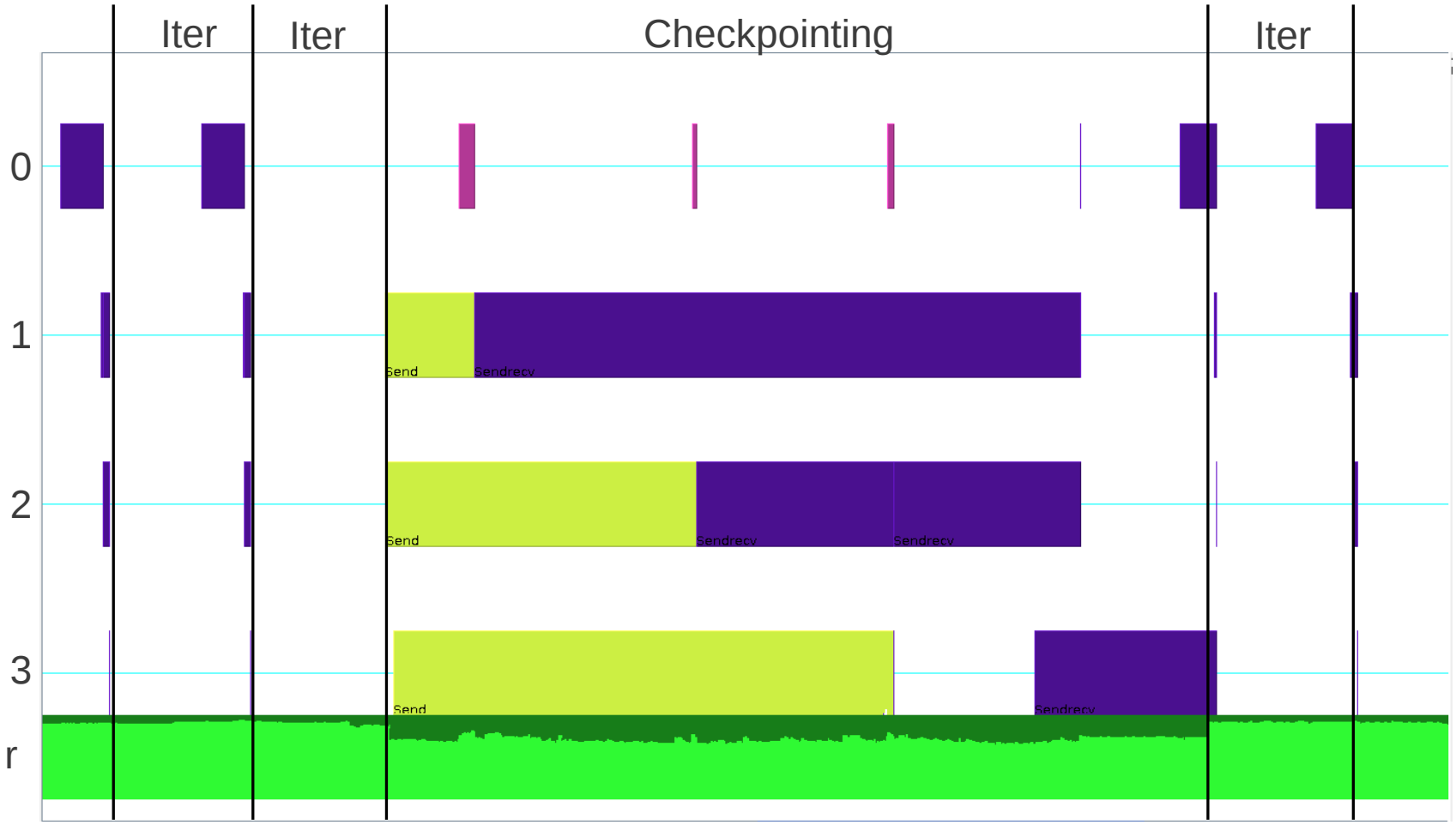

Figure: Tracing MPI activity and node power consumption

Fostering energy efficiency

Controlling hardware states

- Knowing characteristics of the phase(s) allows efficient control.
- Usage of devices and duration of the phase can be estimated.
- Utilize eeClust interface to announce this knowledge.

$$t_{phase} = \frac{E_{change}}{P_{diff}} + t_{change} \quad (1)$$

Phases and active components

Phase bottleneck	I/O activity	Network activity	Potential energy savings
Computation	–	Write-behind to I/O servers	I/O and NIC
Communication	–	–	I/O and CPU
Input/Output	Access data and/or buffer data	Read data if necessary	CPU and NIC

Summary & Conclusions

- CIAO extends the ADIOS interface.
- Named phases indicate high-level semantics.
- Threefold benefit for the user:
  - Performance
  - Efficiency
  - Program analysis
- Monitoring of phase characteristics to steer:
  - I/O behavior
  - Hardware power states

Future Work

- Implementation and evaluation ;-)
- Collaboration with ADIOS developers (and others).