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

1. Introduction
2. ADIOS
3. CIAO interface
4. Benefit for analysis tools
5. Fostering energy efficiency
6. Summary

informatik
die zukunft

Motivation

Conserving 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 OS has little information about future activity.
  - Developers have an idea about the program.

Motivation

Problem of manual annotation

- Convince developers to use the “new” interface is hard.
  - Benefit vs. work.
- Tedious work to annotate
  - Also, the developer must think about future activity.
- Error-prone
  - Sometimes manual annotations are incorrect.

Proposed solution

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

Phases

Phase concept

- Phases span a longer period of execution and achieves a goal.
  - Across multiple function calls, or just a part of a loop.
- Names encode high-level semantics.
  - “Pre-processing”, “Input of topological data”, “iteration”...
- The same name can be used to encode similar behavior.
  - “Exchange-neighbour ghost cells”

Extension of an existing I/O interface

Benefit of the extended interface/library

- Improve knowledge to estimate phase time – optimize I/O.
  - 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.

1. Introduction
2. **ADIOS**
3. CIAO interface
4. Benefit for analysis tools
5. Fostering energy efficiency
6. Summary

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.

Listing 1: Sketched ADIOS code

```
    int NX = 10, NY = 10, NZ = 100;  double matrix[NX][NY][NZ];
    MPI_Comm comm = MPI_COMM_WORLD; int64_t adios_handle;
    int adios_err; uint64_t adios_groupsize, adios_totalsize;

    MPI_Init(&argc, &argv); MPI_Comm_rank(comm, &rank);
    adios_init("example.xml");

    for (t = 0; t < 10 ; t++) {
      adios_start_calculation();
      /* computation */
      adios_stop_calculation();
      /* MPI communication */
      adios_open(&adios_handle, "fullData", "testfile.bp", t == 0 ? "w": "a", &comm);
#include "gwrite_fullData.ch"
      adios_close(adios_handle);
      /* indicate progress for write-behind */
      adios_end_iteration();
    }

    adios_finalize(rank); MPI_Finalize(); return 0;
```

Listing 2: ADIOS example code – gwrite_fullData.ch

```
adios_groupsize = 4 \
                + 4 \
                + 4 \
                + 8 * (NX) * (NY) * (NZ);
adios_group_size (adios_handle, adios_groupsize, &adios_totalsize);
adios_write (adios_handle, "NX", &NX);
adios_write (adios_handle, "NY", &NY);
adios_write (adios_handle, "NZ", &NZ);
adios_write (adios_handle, "matrix_data", matrix);
```

This code is automatically generated from the XML.

Efficient I/O

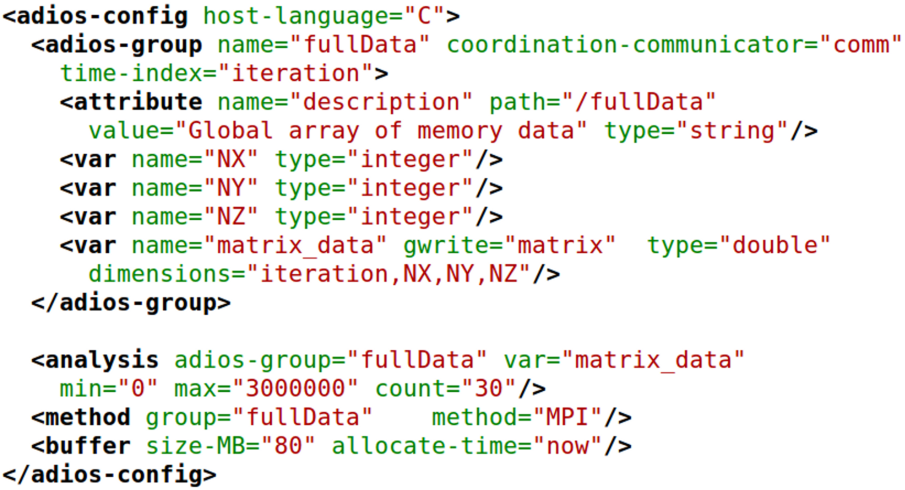
Caching

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

User control in the XML

- Pick the best suitable backend for a supercomputer and task.
- Set optimal parameters such as the cache size.
- Instruct to create derived data (histograms).

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>
```

1 Introduction

2 ADIOS

3 CIAO interface

4 Benefit for analysis tools

5 Fostering energy efficiency

6 Summary

CIAO interface

Extension to ADIOS

- “CIAO” is used to refer to the modified functions.
- Classification into calculation, communication and I/O phases.
  - Add names to phases.
- Goal: Trigger power state and I/O behavior if its advantageous.
  - It is necessary to **predict** future activity!

Listing 3: CIAO example code

```
adios_init("example.xml");

ciao_open(...);
/* read input */
ciao_close(...);

ciao_start_calculation("pre-processing");
/* pre-process input */
ciao_end_calculation();

for (t = 0; t < 10 ; t++) {
  ciao_start_calculation("iteration");
  /* computation */
  ciao_end_calculation();

  ciao_start_communication("exchange-neighbour");
  /* communication */
  ciao_end_communication();

  ciao_open(&adios_handle, "fullData", "testfile.bp", t == 0 ?
       ↪ "w": "a", &comm);
#include "gwrite_fullData.ch"
  ciao_close(adios_handle);
}
adios_finalize(rank);
```

Prediction of future activity

Characterization of phases

- Characterize every named phase:
  - Time, performance (CPU, memory, network utilization).
  - This also enables to classify the phases automatically!

Prediction of phase characteristics

- Characteristics of repeated invocation might be similar.
- Use old characteristics to predict the current phase with:
  - Historic knowledge across program runs.
  - Average (or worst case) characteristics.
- The user can offer hints in the XML to set the predictor.

Prediction of future activity

Estimation of program workflow

- But we want to predict more than just the current phase!
- Sequence of phase transitions could be tracked in CIAO.
  - Probably “iteration-compute” is followed by “exchange-ghost”.
- $\Rightarrow$ predict future phases to estimate future utilization.
- Coarse grained problem of branch prediction.

```
<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>
```

1 Introduction

2 ADIOS

3 CIAO interface

4 Benefit for analysis tools

5 Fostering energy efficiency

6 Summary

Benefit for analysis tools

Phase knowledge enriches profiling and tracing

- Collect an individual profile for each phase.
- Restrict analysis to phases of interest.
- Automatically start tracing/profiling if the phase is interesting.
  - Change in characteristics → interesting.

State of the art

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

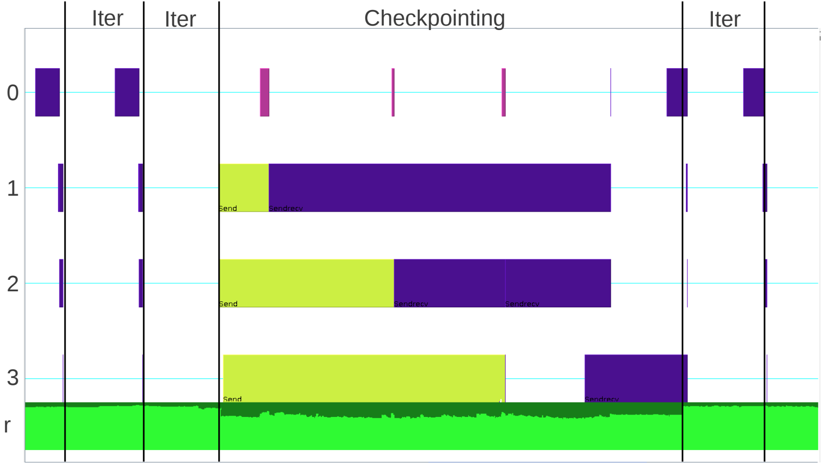


Figure: Tracing MPI activity and node power consumption

1 Introduction

2 ADIOS

3 CIAO interface

4 Benefit for analysis tools

5 Fostering energy efficiency

6 Summary

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.

$$t_{phase} = \frac{E_{change}}{P_{diff}} + t_{change} \tag{1}$$

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

1 Introduction

2 ADIOS

3 CIAO interface

4 Benefit for analysis tools

5 Fostering energy efficiency

6 Summary

Summary & Conclusions

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

Future Work

- Implementation and evaluation of the general concept.
- We seek collaboration to develop/use an open interface!