EXTRACTING MINI-APPS

from HPC software for Total Cost of Ownership optimized system procurement

EXTRACTING MINI-APPS

1 NHR Association

2 Hardware Procurement

3 Total Cost of Ownership

4 Job Mix and Mini-Apps

5 Extraction Pipeline

6 The Apex-Tool

7 Challenges and Outlook

8 Questions

NHR ASSOCIATION

NATIONALES
HOCHLEISTUNGS
RECHNEN

- NHR: „Nationales Hochleistungs Rechnen“

  (National High-Performance Computing)
- An alliance of computing centers
  - Different specializations and hardware installations
  - Common admission process
  - Harmonized computing environment

HARDWARE PROCUREMENT

- What hardware is available
    - The most/best cores, accelerators, memory, storage-system
        - Infeasable for all but the largest computing centers
    - The best we can get in performance per money spent:
        - LINPACK, STREAM, SPEC performance
    - The best we can get in performance per Watt
        - Green 500: LINPACK performance per Watt

- What will a system cost during its lifetime?
    - Keep users in mind during procurement

TOTAL COST OF OWNERSHIP

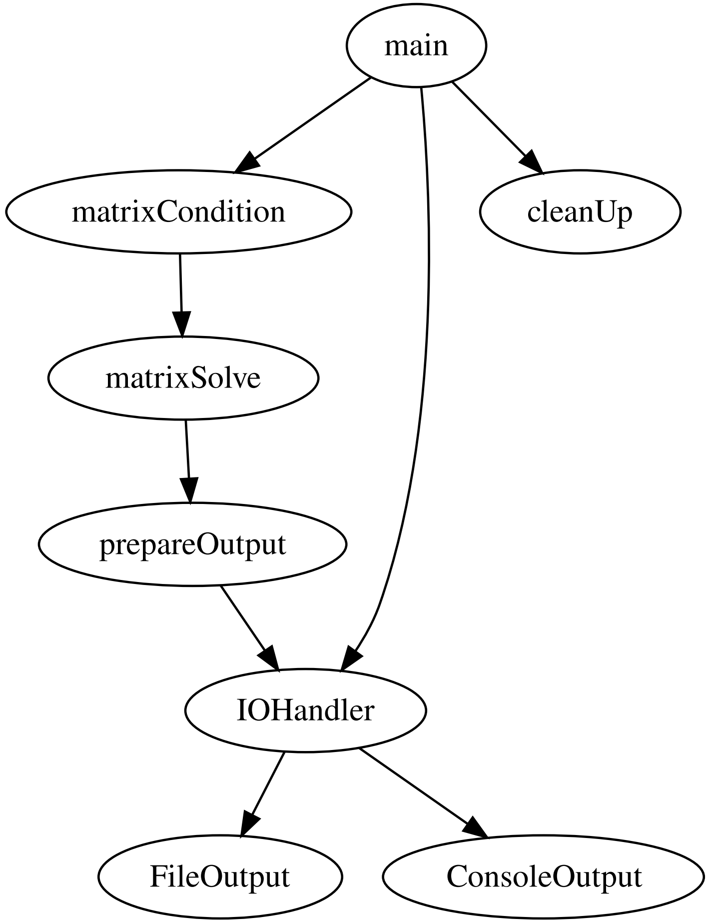
- Score procurement offers not only on performance
- Model total cost of ownership of a system with:
  - Hard-/Software investment costs
  - Cooling cost and environmental impact
  - Technical and administrative staff
  - **Power consumption (of a job mix)**

JOB MIX AND MINI-APPS

- Job mix is a user-dependent metric
  - What is the system actually being used for?
- What do these jobs benefit from the most?
  - Physics Simulation → CPU
  - Big Data → Storage / Memory
  - AI → Accelerators
- Monitor the usage and translate the jobmix into procurement criteria
  - „Let them run LAMMPS, GROMACS and OpenFoam“
  - „Let them run that one a.out executable, if they can figure out how to get it to run“

JOB MIX AND MINI-APPS

- Scientic and HPC applications are:
  - Large
  - Complex
  - Different code/software patterns
  - **Representative workload**



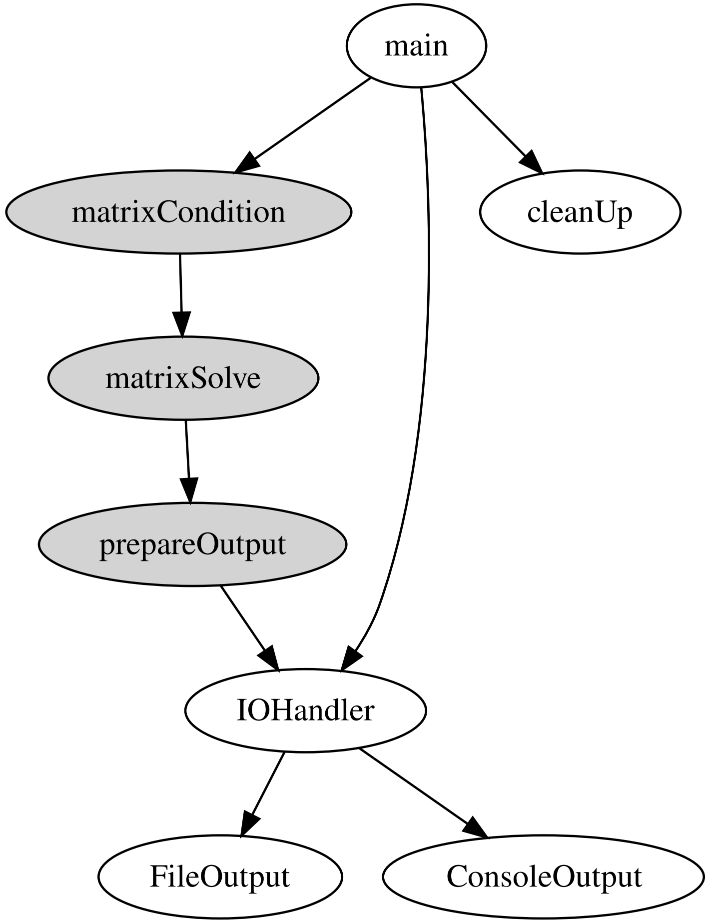
JOB MIX AND MINI-APPS

- Scientic and HPC applications are:
  - Large
  - Complex
  - Different code/software patterns
  - **Representative workload**

- Use a Mini-App instead! [1]

→ shrink size but keep “characteristic”

→ “characteristic” e.g. computational kernel



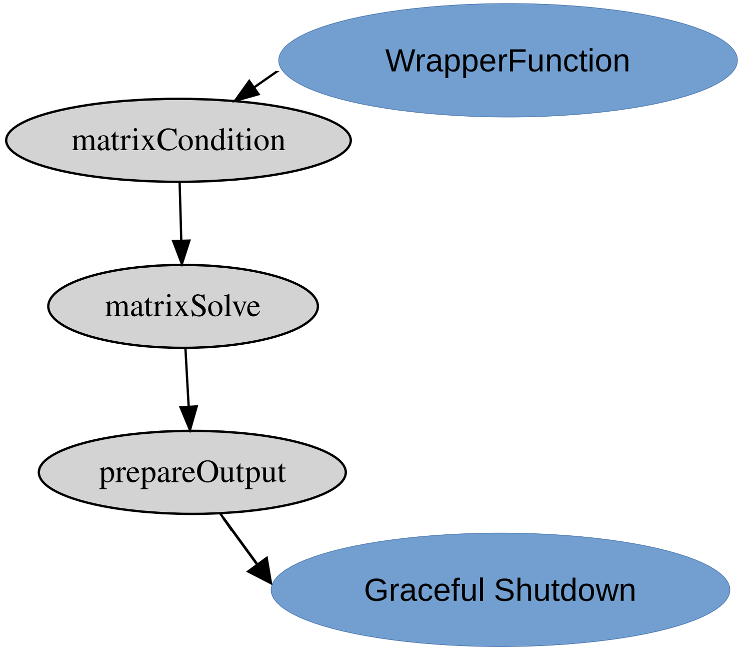
JOB MIX AND MINI-APPS

- Scientic and HPC applications are:
  - Large
  - Complex
  - Different code/software patterns
  - **Representative workload**

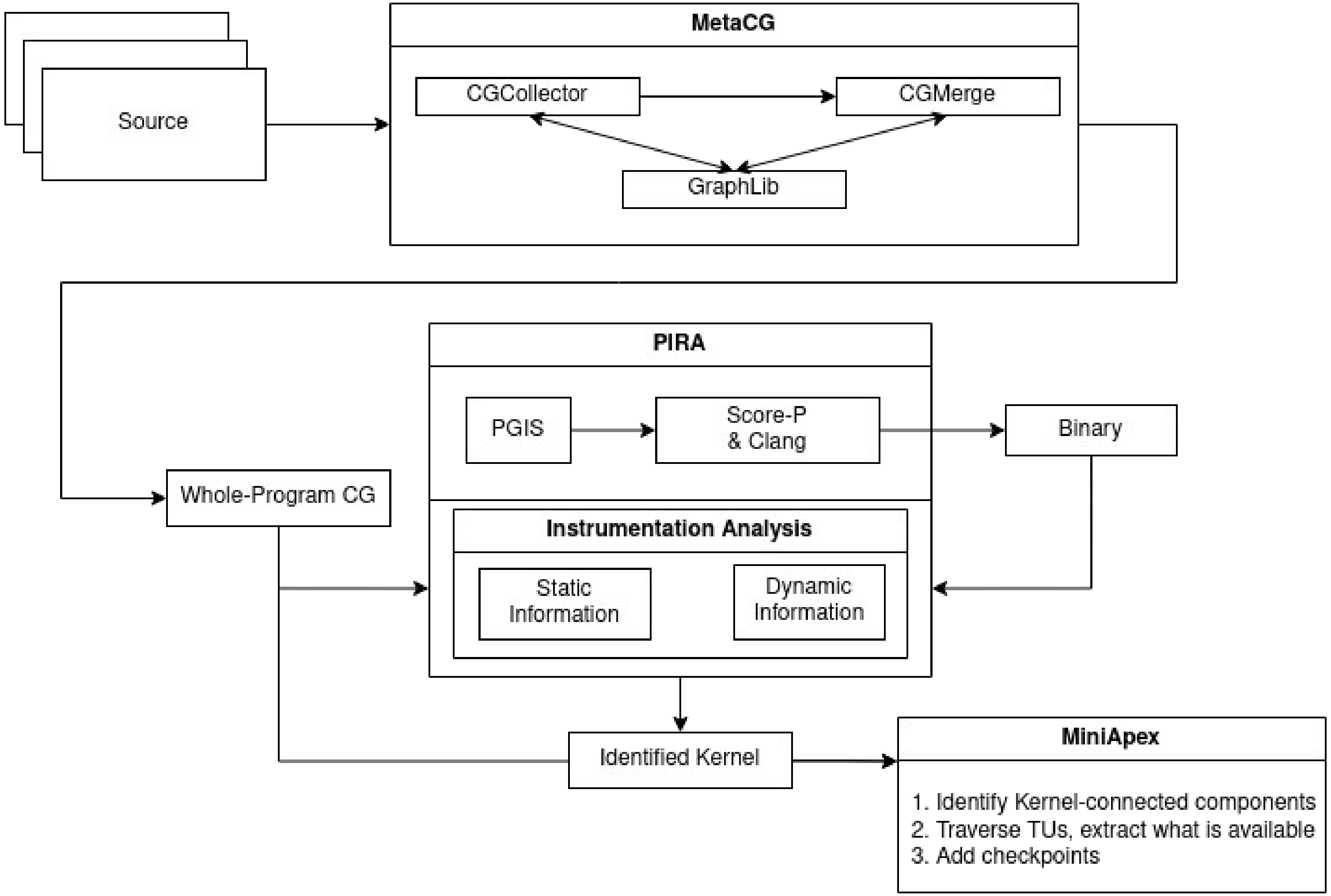
- Use a Mini-App instead! [1]

→ shrink size but keep “characteristic”

→ “characteristic” e.g. computational kernel



EXTRACTION PIPELINE



Analyze the program-structure [2]

Profile for the kernel [3]

Extract the Kernel [4]

THE APEX-TOOL

- Is a Clang-frontend based compiler-tool to do source manipulation
  - Queries the AST
    - AST: Abstract Syntax Tree
      - holds most program information

```
//helper.h
struct S {
  int i;
};
void printS(S s)
```

```
#include "helper.h"
int main(){
  S s;
  s.i=5;
  printS(s);
}
```

```
//helper.cpp
#include <iostream>
void printS(S s){ std::cout<<s.i<<"\n";}
```

```
TranslationUnitDecl
|-RecordDecl struct S definition
| `-FieldDecl i 'int'
|-FunctionDecl printS 'void (S)'
| `-ParmVarDecl s 'S':'S'
`-FunctionDecl main 'int ()'
  `-CompoundStmt
    |`-VarDecl s 'S':'S'
    |`-BinaryOperator '='
    | |`-DeclRefExpr 'S':'S' Var 's' 'S':'S'
    | |`-IntegerLiteral 5
    `-CallExpr 'void'
      |`-DeclRefExpr 'printS' 'void (S)'
```

THE APEX-TOOL: BASICS

1) Given the kernel, we must identify the call subtree
   - This is done via the whole-program call-graph
2) Find all functions we use and have defined
   - The AST can not provide a definition
     - printS is only declared in the main file (different *.cpp)

➜Get as text block

3) Find all accessed globals
   - The AST has this information
     - We have the definition of **struct S** (*.h is included)

➜Get as text Block

4) Find all #include statements ...

```
//helper.h
struct S {
  int i;
};
void printS(S s)

#include "helper.h"
int main(){
  S s;
  s.i=5;
  printS(s);
}

//helper.cpp
#include <iostream>
void printS(S s){ std::cout<<s.i<<"\n";}
```

THE APEX-TOOL: ADVANCED

4) The #include statements are handled by the preprocessor
   - This is also true for #defines, #if[n]defs, #pragmas
     - All resolved before we build the AST
   - Write preprocessor hooks to extract the information
     - Not context sensitive
   - insert the #includes #defines, #if[n]defs, #pragmas
     - need to map context insensitive preprocessor information to context sensitive AST information
       - Only hint are source file locations

```
//helper.h
struct S {
  int i;
};
void printS(S s)
```

```
#include "helper.h"
int main(){
  S s;
  s.i=5;
  printS(s);
}
```

```
//helper.cpp
#include <iostream>
void printS(S s){ std::cout<<s.i<<"\n";}
```

THE APEX-TOOL: ADVANCED

- With the Preprocessor we know:
  - Line 1-3 #if _OPENMP …
  - Line 2 #include …
  - Line 6-13 #if _OPENMP …
  - Line 7-9 #if USE_MPI …
  - Line 15 #pragma …
- With the function extraction we know:
  - Line 5-20 void IntegrateStressForElems([...])

```
//Excerpt of Lulesh code
 1  #if _OPENMP
 2  # include <omp.h>
 3  #endif
 4  [...]
 5  void IntegrateStressForElems([...]){
 6  #if _OPENMP
 7  #if USE_MPI
 8      int a=5;
 9  #endif
10      Index_t numthreads = omp_get_max_threads();
11  #else
12      Index_t numthreads = 1;
13  #endif
14  [...]
15  #pragma omp parallel for firstprivate(numElem)
16    for( Index_t k=0 ; k<numElem ; ++k ){[...]}
17  [...]
```

TECHNISCHE
UNIVERSITÄT
DARMSTADT

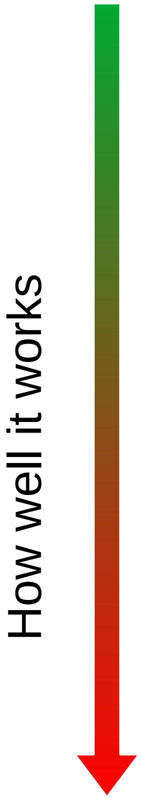
SCIENTIFIC
COMPUTING

THE APEX-TOOL: ADVANCED

- With the Preprocessor we know:
  - Line 1-3 #if _OPENMP …
  - Line 2 #include …
  - Line 6-13 #if _OPENMP …
  - Line 7-9 #if USE_MPI …
  - Line 15 #pragma …
- With the function extraction we know:
  - Line 5-20 void IntegrateStressForElems([…])

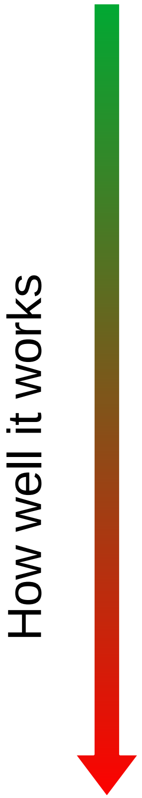
```
//Excerpt of Lulesh code
 1  #if _OPENMP
 2  # include <omp.h>
 3  #endif
 4  [...]
 5  void IntegrateStressForElems([...]){
 6  #if _OPENMP
 7  #if USE_MPI
 8      int a=5;
 9  #endif
10      Index_t numthreads = omp_get_max_threads();
11  #else
12      Index_t numthreads = 1;
13  #endif
14  [...]
15  #pragma omp parallel for firstprivate(numElem)
16    for( Index_t k=0 ; k<numElem ; ++k ){[...]}
17  [...]
```

CHALLENGES AND OUTLOOK



- Single translation unit C-code
- Multi translation unit C-code
- Code with C++ components (new, delete, classes)
- Templates
- Complex class inheritance and polymorphism
- Checkpointing
  - Nested arrays
  - Private class members
  - Multi level pointer structures
- Every code ever written in C++

CHALLENGES AND OUTLOOK



- Single translation unit C-code
- Multi translation unit C-code
- Code with C++ components (new, delete, classes)
- Templates
- Complex class inheritance and polymorphism
- Checkpointing
  - Nested arrays
  - Private class members
  - Multi level pointer structures
- Every code ever written in C++

Even in its current state it can be helpful:

- Tool-assistend mini-app extraction with manual work (mostly wrapper)
- Extracting small components for manual optimization
  - should lead to simplified reintegration of changes

If you know of an HPC code, that is C/C++ and has a small kernel compared to its total code size:

Please tell me!

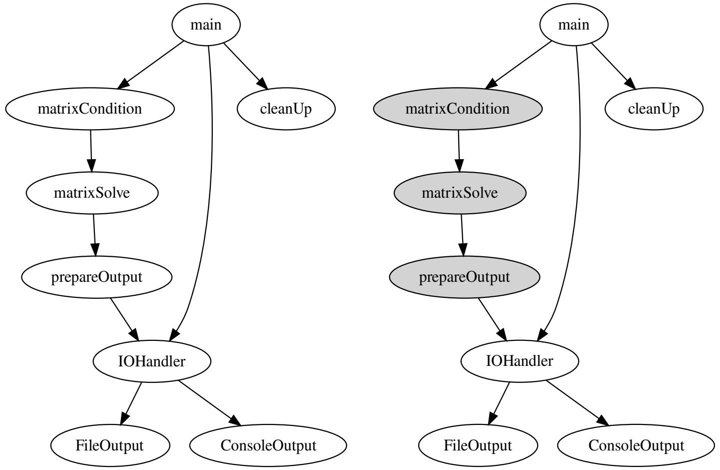
TECHNISCHE
UNIVERSITÄT
DARMSTADT

SCIENTIFIC
COMPUTING

$$Cost_{HW} = \frac{jobmix[coreh]}{lifetime[coreh]} * \frac{nodesjobmix}{nodestotal} * \frac{cost_{offer}}{uptime_{projected}}$$

THANK YOU FOR YOUR ATTENTION

```
//Excerpt of Lulesh code
#if _OPENMP
# include <omp.h>
#endif
[...]
void IntegrateStressForElems([...]){
#if _OPENMP
#if USE_MPI
    int a=5;
#endif
    Index_t numthreads = omp_get_max_threads();
#else
    Index_t numthreads = 1;
#endif
[...]
#pragma omp parallel for firstprivate(numElem)
  for( Index_t k=0 ; k<numElem ; ++k ){[...]}
[...]
```



REFERENCES

[1]	*From Valid Meassurements to Miniapps*	by Jan-Patrick Lehr	doi:10.26083/tuprints-00020943
[2]	PIRA/PGIS	https://github.com/tudasc/PIRA	doi:10.1145/3281070.3281071
[3]	MetaCG	https://github.com/tudasc/MetaCG	doi:10.1145/3427764.3428320
[4]	CTUApex	https://git.rwth-aachen.de/tim.heldmann/CTUApex	