EXTRACTING MINI-APPS

from HPC software for Total Cost of Ownership optimized system procurement
# EXTRACTING MINI-APPS

<table>
<thead>
<tr>
<th></th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NHR Association</td>
</tr>
<tr>
<td>2</td>
<td>Hardware Procurement</td>
</tr>
<tr>
<td>3</td>
<td>Total Cost of Ownership</td>
</tr>
<tr>
<td>4</td>
<td>Job Mix and Mini-Apps</td>
</tr>
<tr>
<td>5</td>
<td>Extraction Pipeline</td>
</tr>
<tr>
<td>6</td>
<td>The Apex-Tool</td>
</tr>
<tr>
<td>7</td>
<td>Challenges and Outlook</td>
</tr>
<tr>
<td>8</td>
<td>Questions</td>
</tr>
</tbody>
</table>
NHR ASSOCIATION

- 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

- Scientific and HPC applications are:
  - Large
  - Complex
  - Different code/software patterns
  - Representative workload
JOB MIX AND MINI-APPS

- Scientific 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

- Scientific 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]

1. Identify Kernel-connected components
2. Traverse TUs, extract what is available
3. Add checkpoints
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
#include "helper.h"
struct S {
  int i;
};
void printS(S s) {
  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

```c
//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  [...]

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!
THANK YOU FOR YOUR ATTENTION

//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 #pragma omp parallel for firstprivate(numElem)
9 for (Index_t k=0 ; k<numElem ; ++k) {...}
10 #else
11 int a=5;
12 #endif
13 #endif

REFERENCES

[1] From Valid Meassurements to Miniapps
   by Jan-Patrick Lehr  doi:10.26083/tuprints-00020943

[2] PIRA/PGIS


[4] CTUApepx
   https://git.rwth-aachen.de/tim.heldmann/CTUApepx