An Empirical Study of Computation-Intensive Loops for Identifying and Classifying Loop Kernels

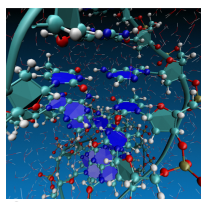
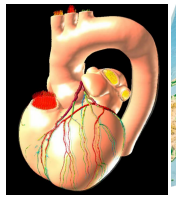
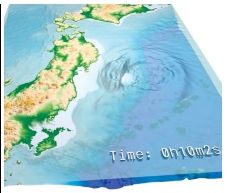
Masatomo Hashimoto
Masaaki Terai
Toshiyuki Maeda
Kazuo Minami

Agenda

- Performance engineering of scientific applications
  - Identifying source code region (loops) that we should optimize
  - Estimating expected performance of the loops to set the goal of performance tuning
    - Classifying loops according to expected performance
- Empirical study of 1020 real applications
  - How many loops should we optimize?
  - What efficiency can we expect from the loops?

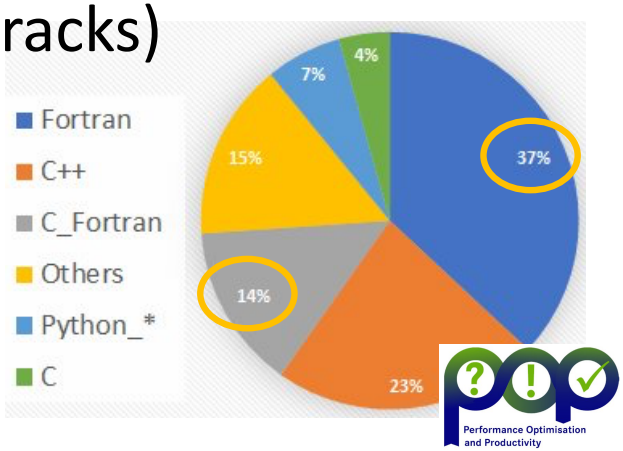
Large-Scale Scientific Applications

- Simulate real-world phenomena
- Massively parallel
- Running on supercomputers
  - E.g. the K computer
    - 11.28 PFLOPS peak performance
    - 705,024 SPARC64 2GHz cores (864 racks)
    - 1.41 PiB memory (2GiB/core)
    - 10GiBx2 bandwidth interconnect
- Written in **Fortran**, C++, …

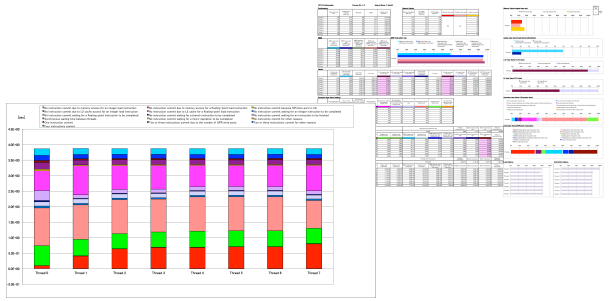
©NIMS&UCL ©UT-Heart ©U-Tokyo

©Fujitsu & RIKEN AICS



Improving Performance of Applications

- Essential for scientific progress
  - More efficiency, more scientific results
    - Simulated time/resolution, accuracy of model
- Requires human intervention
  - Identifying loops to be tuned
    - *Computational kernel*



  - Changing compiler opts and/or transforming loops

Computational Kernel

- (Nested) loop(s) in a subprogram
  - Array references + floating-point operations
- E.g.

```
integer m, n1, n2, n3, i1, i2, i3
double precision u(n1, n2, n3), v(n1, n2, n3), r(n1, n2, n3), a(0:3), u1(m), u2(m)
do i3=2, n3−1
 do i2=2, n2−1
  do i1=1, n1
   u1(i1) = u(i1, i2−1,i3) + u(i1, i2+1, i3) + u(i1, i2 , i3−1) + u(i1, i2 , i3+1)
   u2(i1) = u(i1, i2−1,i3−1) + u(i1, i2+1, i3−1) + u(i1, i2−1, i3+1) + u(i1, i2+1, i3+1)
  enddo
  do i1=2,n1−1
   r(i1, i2, i3) = v(i1, i2, i3) − a(0) * u(i1, i2 ,i3)&
                                 − a(2) * ( u2(i1) + u1(i1−1) + u1(i1+1) )&
                                 − a(3) * ( u2(i1 – 1) + u2(i1 + 1) )

  enddo
 enddo
enddo
```

From Multi-Grid (NAS Parallel Benchmark)

Hurdles in Performance Tuning

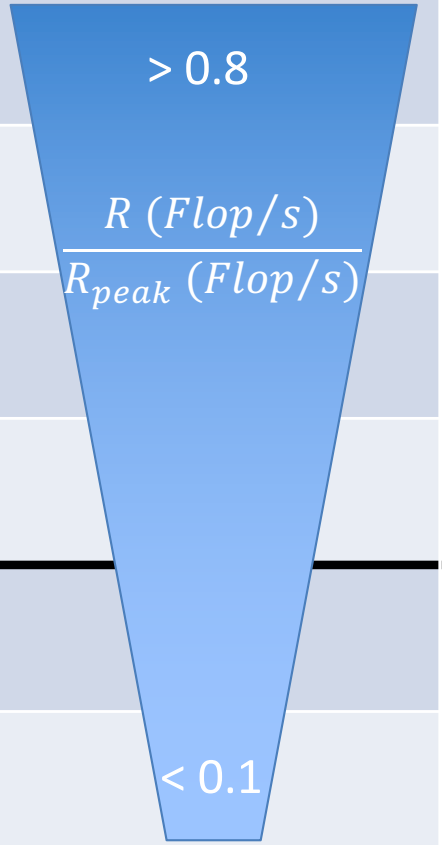
- Apps are programmed by *scientists* and tuned by *performance engineers*
  - Comprehending scientists' code is painful
    - Identification of algorithms and kernels
- Combinatorial explosion of search space
  - Compiler options
    - E.g. GCC has 100 performance related options
  - Loop transformations and their parameters
    - E.g. unrolling, blocking, fusion, fission, …

Learning from Performance Engineers

- Predicting good loop transformation
  - Studied in our previous work for given computational kernels [MSR2015]
- Classification of loops
  - Kernel or non-kernel
  - Six kernel classes (groups of similar kernels)
    - For estimating target performance/efficiency
      - Similar kernel, similar efficiency

Kernel Classes

Class	Traits	Typical Applications	Efficiency
M1	Matrix-matrix multiplication	Density functional theory	
M2	Simple loop body	Molecular dynamics	
M3	Simple loop body + stencil	Particular high-order accuracy stencil	
M4	Complex loop body	Climate model (physical), plasma (particle-in-cell)	
M5	Stencil	Climate model (dynamical), fluid dynamics, earthquake	
M6	Stencil (indirect array reference)	Fluid dynamics (finite element method)	



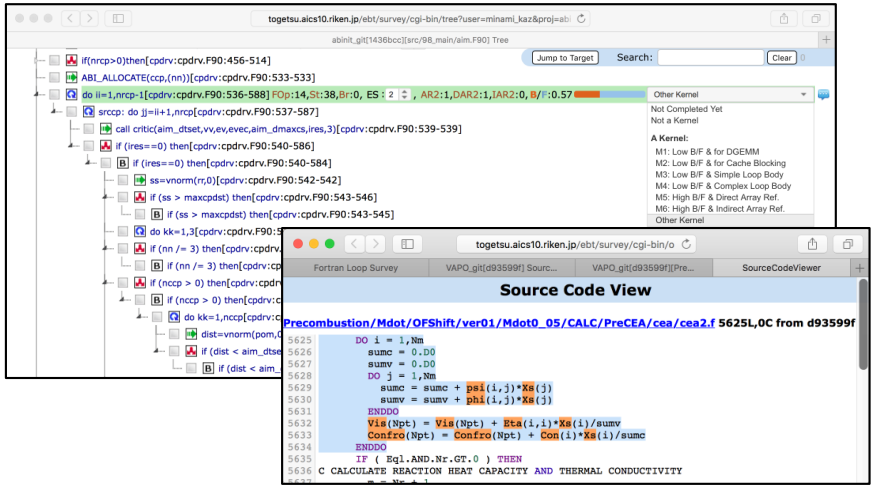
(SPARC64VIIIfx)

Learning Kernel Classification from Performance Engineers

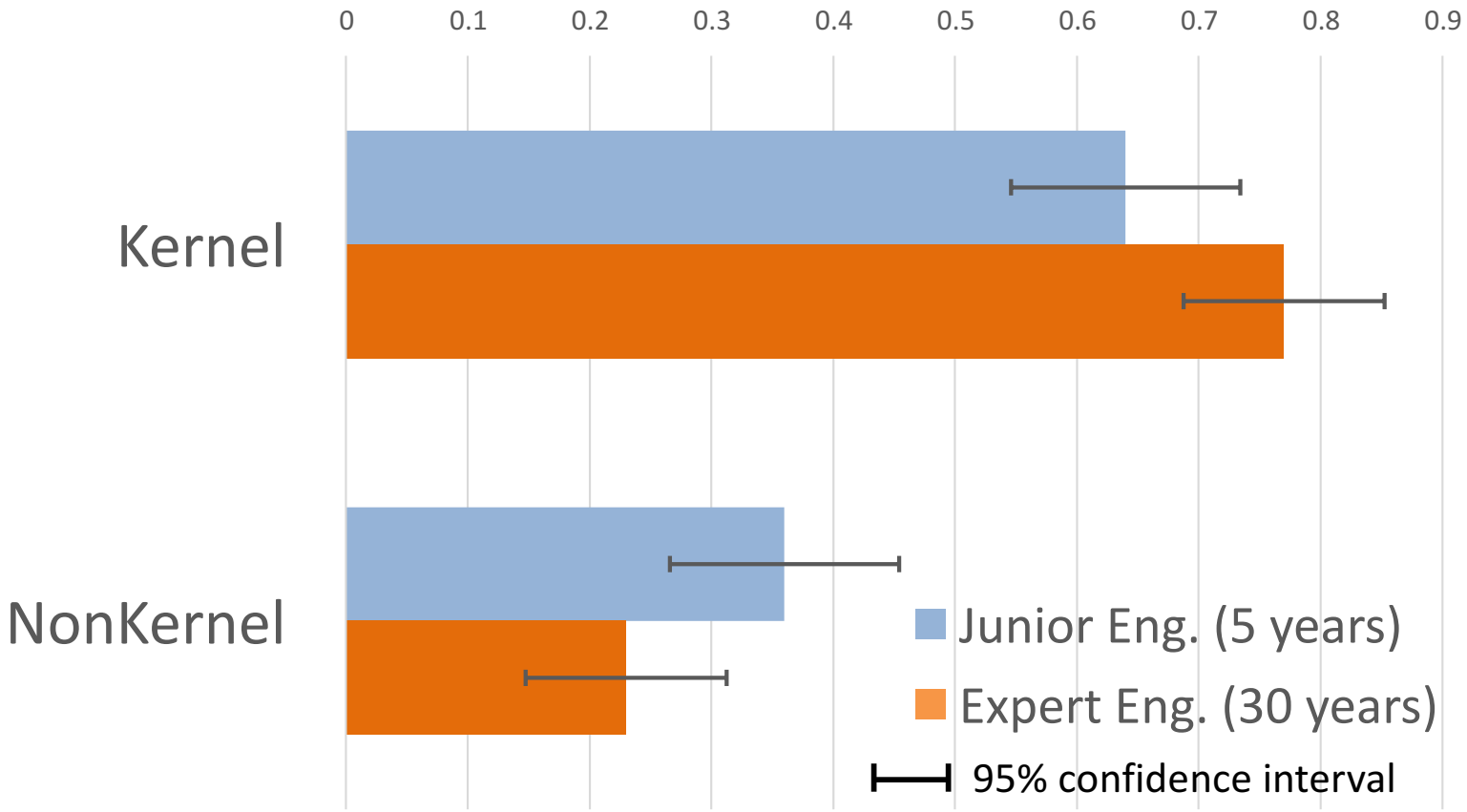
- Selected 175,963 loops from 1020 Fortran applications on GitHub
  - Use of OpenMP, MPI, and/or OpenACC
- Manually classified 100 loops that are randomly sampled from the loops
  - By experienced performance engineers
    - Code comprehension assistance
- Statistically estimated distribution of classes over the 175K loops
- Created predictive models based on the manual classification results

CCA: Code Comprehension Assistance

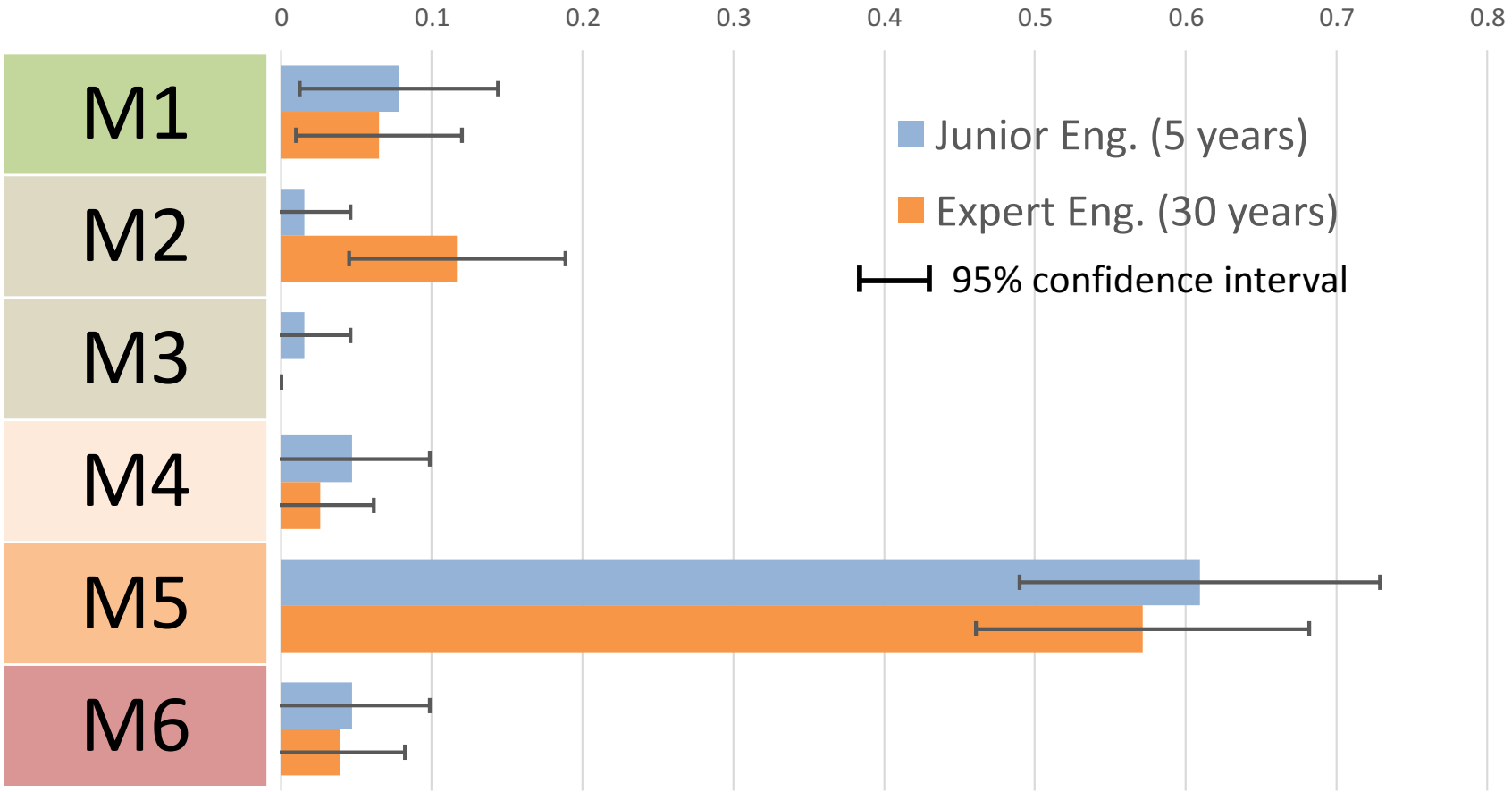
- Fortran parser
  - Standards: F77, F90, F95, F2003, F2008
  - Vendor-specific extensions/directives
    - Fujitsu, GNU, IBM, Intel, PGI, OpenMP, OpenACC
  - Out-of-order parsing
    - No need to hook build process
- Outline view of call-tree
  - With code metrics
- Topic analysis of source code
- Source code viewer
- Available at https://github.com/ebt-hpc/cca



Distribution of Loops



Distribution of Kernel Classes



Predicting Kernel Class of a Loop

- Constructing predictive model
  - Feature vector of loop → kernel class (or non-kernel)
    - Machine learning algo.: SVC (Support Vector Classification)
    - Feature vector: ***static*** loop metrics
      - E.g. num of calls, num of indirect array refs
    - Training data set: manual classification result (100 samples)
- Predicting whether a loop is a kernel or not
  - Classification accuracy: 0.81 (20-fold cross validation)
- Predicting whether a kernel is M5 or not
  - Classification accuracy: 0.94 (20-fold cross validation)

Summary

- Performance tuning of scientific applications
  - Essential and demanding
- Learning from performance tuning experts
  - Classification of loop kernels into six classes
    - Estimating target performance/efficiency
- Estimating distribution of kernel classes
  - 175K loops from 1K Fortran applications on GitHub
    - 70-85% are *kernels*
    - 35-55% are *M5 kernels* (stencil, <15% expected efficiency)
- Correlation between static loop features and kernel classes
  - Machine learning (SVC) + training set (100 sampled loops)
    - Classification accuracy 0.81 (Is given loop a kernel or not?)
    - Classification accuracy 0.94 (Is given kernel M5 or not?)

THANK YOU FOR YOUR ATTENTION

- Raw results of the experiments
  - https://github.com/ebt-hpc/icpe2017
- Outlines of 1020 applications
  - https://ngse.riken.jp/outline
- Our tool (CCA/EBT)
  - https://github.com/ebt-hpc/cca