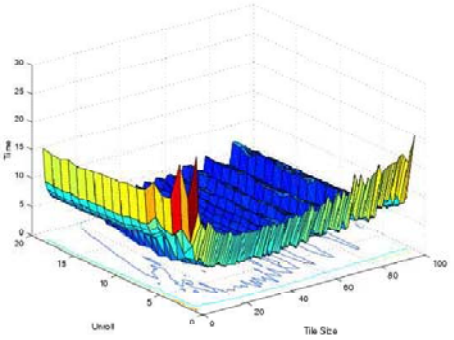
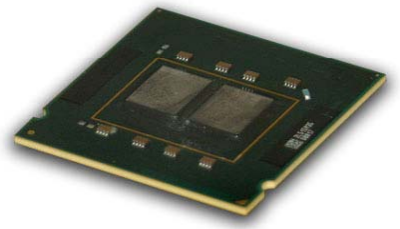


Continuous Iterative Adaptive Compilation

Grigori Fursin

Alchemy group, INRIA Futurs, France



My background

- Ph.D. degree from Edinburgh University, UK *(1999-2004)*

  Program iterative optimizations and performance prediction

- Postdoctoral researcher at INRIA Futurs, France *(2004-2007)*
- Tenured research scientist at INRIA Futurs, France *(2007 …)*

  Iterative compilation, run-time adaptation,
  machine learning, Design Space Exploration

  (Grants and funding from HiPEAC European network of excellence, MilePost and SARC projects, INRIA)

Outline

- ALCHEMY group overview
- Motivation
- Prerequisites for my work (iterative compilation)
- Research plan
- Past and current research
- Future work
- Other Alchemy group activities
- Collaborations, public software and dissemination

Alchemy group overview

Architectures, Languages and Compilers to Harness the End of Moore Years

- *Architecture & Compiler research group*
  - 7 Faculty:
    - Hugues Berry (CR)
    - Albert Cohen (DR)
    - Christine Eisenbeis (DR)
    - Grigori Fursin (CR)
    - Olivier Temam (DR)
    - Cedric Bastoul (Assist. Prof.)
    - Frederic Gruau (Assist. Prof.)
  - 2 Engineers:
    - Sylvain Girbal
    - Cupertino Miranda
  - 13 PhD students
- *High-Performance general-purpose & embedded architectures*
- *Currently participating to 6 European and 3 French projects*

Alchemy group overview

- *Topics:*
  - Iterative optimization
  - Influence of datasets
  - Machine learning
  - Run-time continuous optimizations
  - GCC as a research platform
  - Manual program tuning (decision tree)
  - Programming models
  - Multi-Core architectures
  - Streaming applications/architectures
  - Simulation (modular simulation environment)
  - Architecture design space exploration
  - Bio-inspired systems (blob computing)

More information:
https://alchemy.futurs.inria.fr

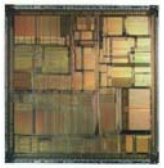
Motivation

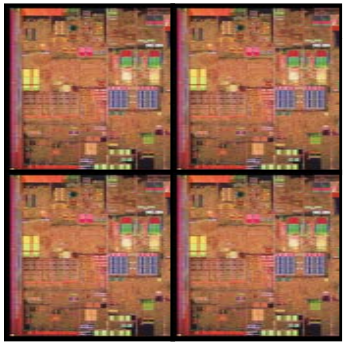
Current innovations in science and industry demand ever-increasing computing resources while placing strict requirements on ***system performance, power consumption, size, response, reliability, portability and design time***.

Motivation

Current innovations in science and industry demand ever-increasing computing resources while placing strict requirements on ***system performance, power consumption, size, response, reliability, portability and design time***.

High-performance computing systems tend to evolve toward
complex heterogeneous multi-core systems



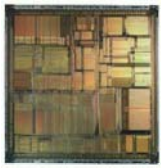


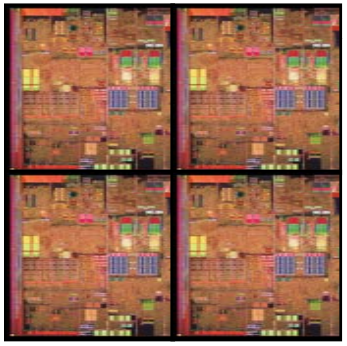
dramatically increased optimization time

Motivation

Current innovations in science and industry demand ever-increasing computing resources while placing strict requirements on ***system performance, power consumption, size, response, reliability, portability and design time***.

High-performance computing systems tend to evolve toward
complex heterogeneous multi-core systems





dramatically increased optimization time

Optimizing compilers play a key role in ***producing executable codes quickly and automatically*** while satisfying all the above requirements for a broad range of programs and architectures.

Motivation

Current state-of-the-art compilers and optimizers fail to deliver best performance on modern systems due to

- simplistic hardware models for rapidly evolving hardware
- fixed black-box optimization heuristics and inability to fine-tune applications
- inability to reuse optimization knowledge among different programs and architectures
- lack of run-time information and inability to adapt to varying program and system behavior at run-time with low overhead

Motivation

Current state-of-the-art compilers and optimizers fail to deliver best performance on modern systems due to

- simplistic hardware models for rapidly evolving hardware
- fixed black-box optimization heuristics and inability to fine-tune applications
- inability to reuse optimization knowledge among different programs and architectures
- lack of run-time information and inability to adapt to varying program and system behavior at run-time with low overhead

Current compiler and optimization technologies should be revisited to keep pace with rapidly evolving hardware

Motivation

Current state-of-the-art compilers and optimizers fail to deliver best performance on modern systems due to

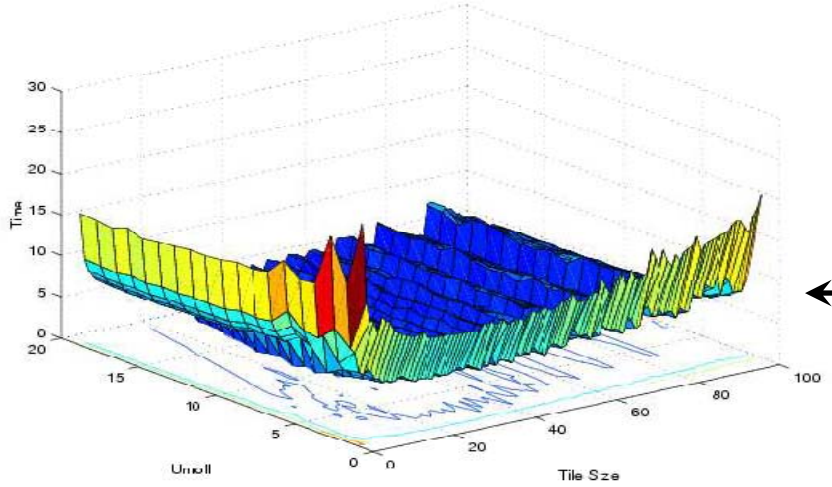
- simplistic hardware models for rapidly evolving hardware
- fixed black-box optimization heuristics and inability to fine-tune applications
- inability to reuse optimization knowledge among different programs and architectures
- lack of run-time information and inability to adapt to varying program and system behavior at run-time with low overhead

Current compiler and optimization technologies should be revisited to keep pace with rapidly evolving hardware

Need static compilers that can continuously and automatically learn how to optimize programs, and have an ability to adapt at run-time for different behavior and constraints

Iterative optimizations

Optimization spaces (set of all possible program transformations) are large, non-linear with many local minima



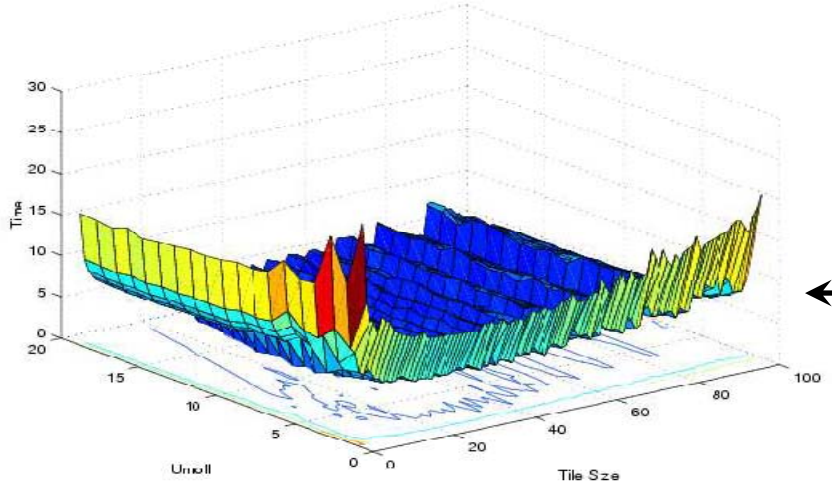
Finding a good solution may be long and non-trivial

matmul, 2 transformations, search space = 2000

swim, 3 transformations, search space = 10^{52}

Iterative optimizations

Optimization spaces (set of all possible program transformations) are large, non-linear with many local minima



Finding a good solution may be long and non-trivial

matmul, 2 transformations, search space = 2000

swim, 3 transformations, search space = 10^{52}

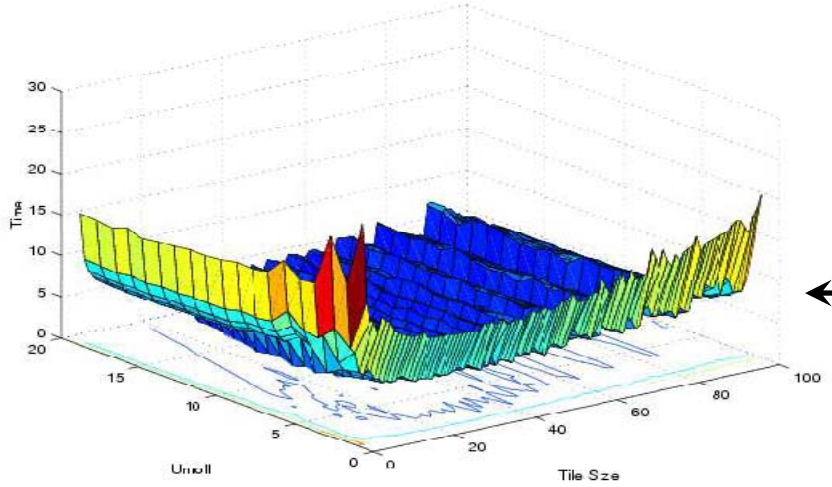
Recent technique - iterative compilation: learn program behavior across executions

High potential (O'Boyle, Cooper), but:

- slow
- the same dataset is used
- no run-time adaptation
- no optimization knowledge reuse

Iterative optimizations

Optimization spaces (set of all possible program transformations) are large, non-linear with many local minima



Finding a good solution may be long and non-trivial

matmul, 2 transformations, search space = 2000

swim, 3 transformations, search space = 10^{52}

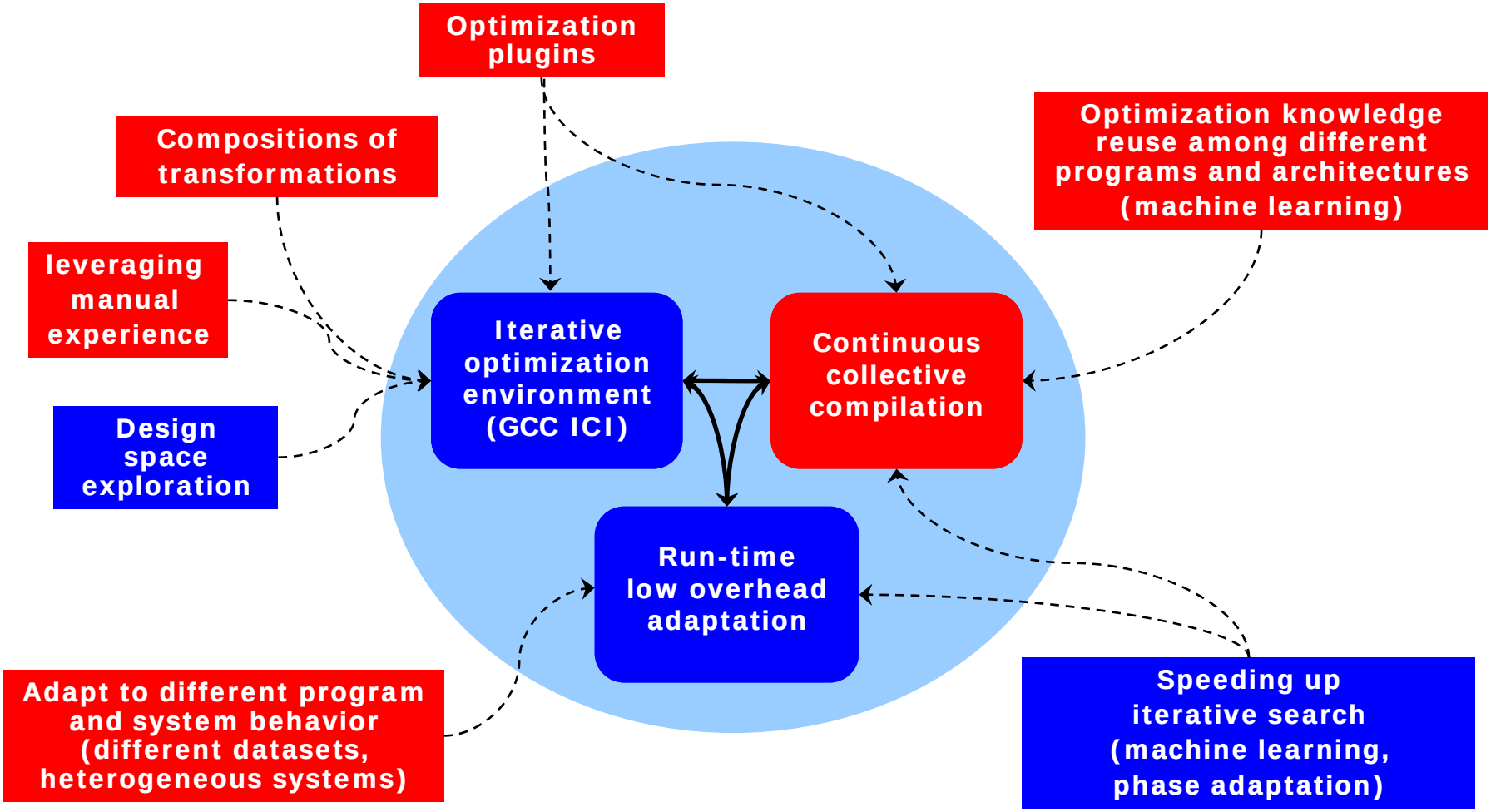
Recent technique - iterative compilation: learn program behavior across executions

High potential (O'Boyle, Cooper), but:

- slow
- the same dataset is used
- no run-time adaptation
- no optimization knowledge reuse

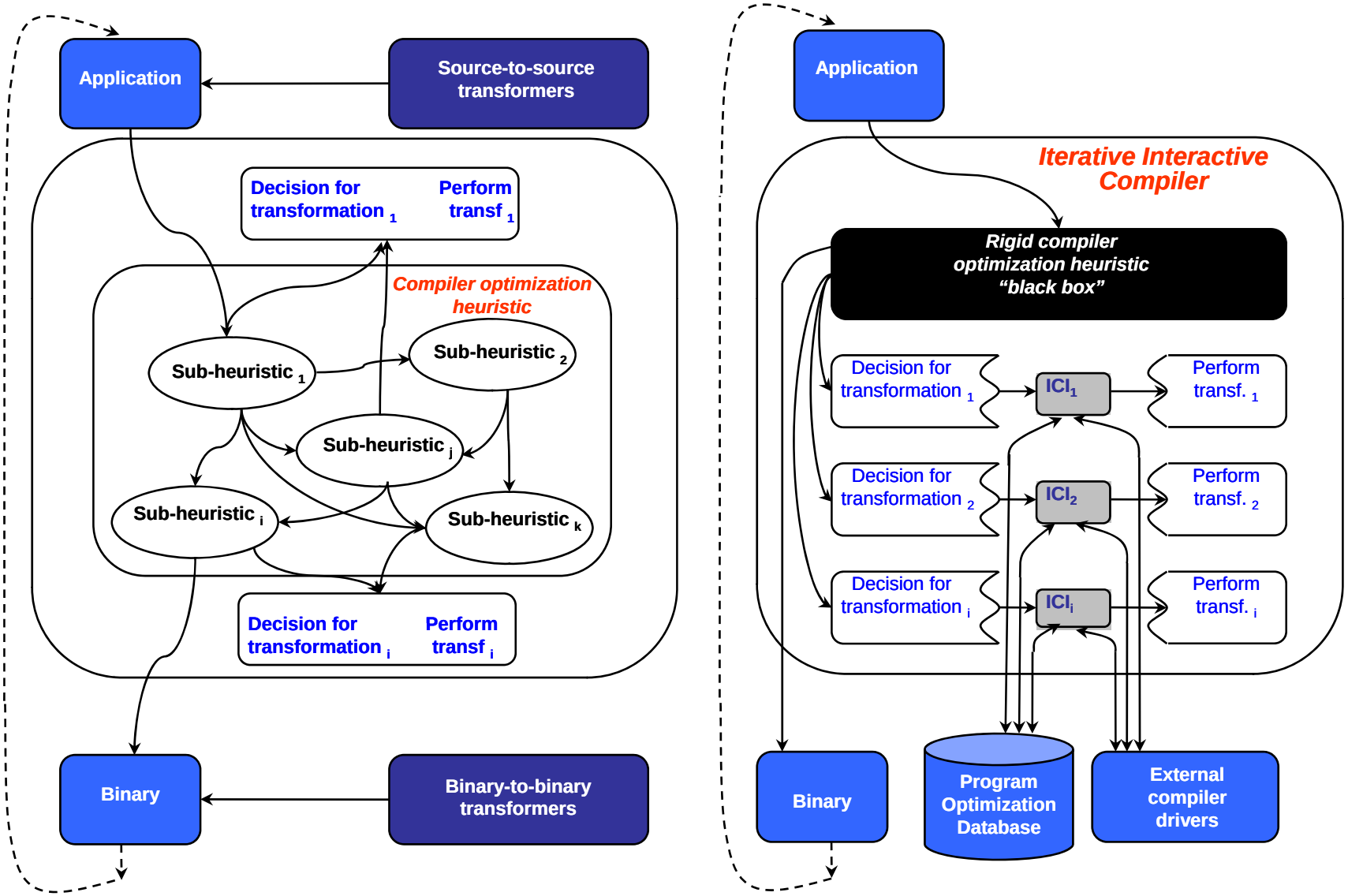
Solving these problems is non-trivial

Research activities

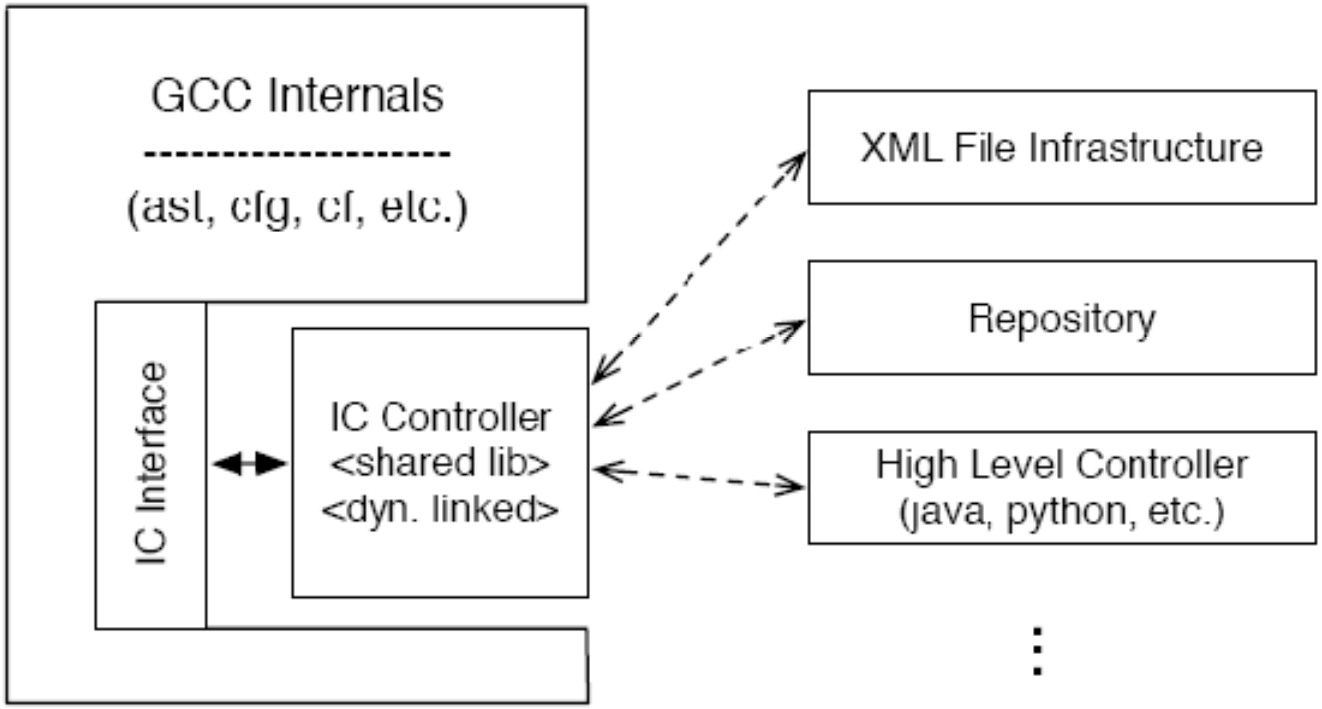


Iterative compilation

Interactive Compilation Interface (ICI)



New Interactive Compilation Interface



- Pass manager plug-in implemented to enable pass reordering
- Static features extraction implemented
- Connecting to the machine learning tool to predict best optimization order
- Release in August/September 2007

Collaboration with IBM Haifa

Iterative compilation

- G.G.Fursin, M.F.P.O'Boyle, and P.M.W. Knijnenburg. Evaluating Iterative Compilation. *Proceedings of the 15th Workshop on Languages and Compilers for Parallel Computing* (LCPC'02), College Park, MD, USA, pages 305-315, 2002

- Grigori Fursin. Iterative Compilation and Performance Prediction for Numerical Applications. Ph.D. thesis, University of Edinburgh, Edinburgh, UK, January 2004

- Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. Fast and Accurate Method for Determining a Lower Bound on Execution Time. *Concurrency Practice and Experience*, 16(2-3), pages 271-292, 2004

- Shun Long and Grigori Fursin. A heuristic search algorithm based on Unified Transformation Framework. *Proceedings of the 7th International Workshop on High Performance Scientific and Engineering Computing (HPSEC-05)*, pages 137-144, Oslo, Norway, June 2005

- Grigori Fursin and Albert Cohen. Building a Practical Iterative Interactive Compiler. *1st International Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'07),* Ghent, Belgium, January 2007

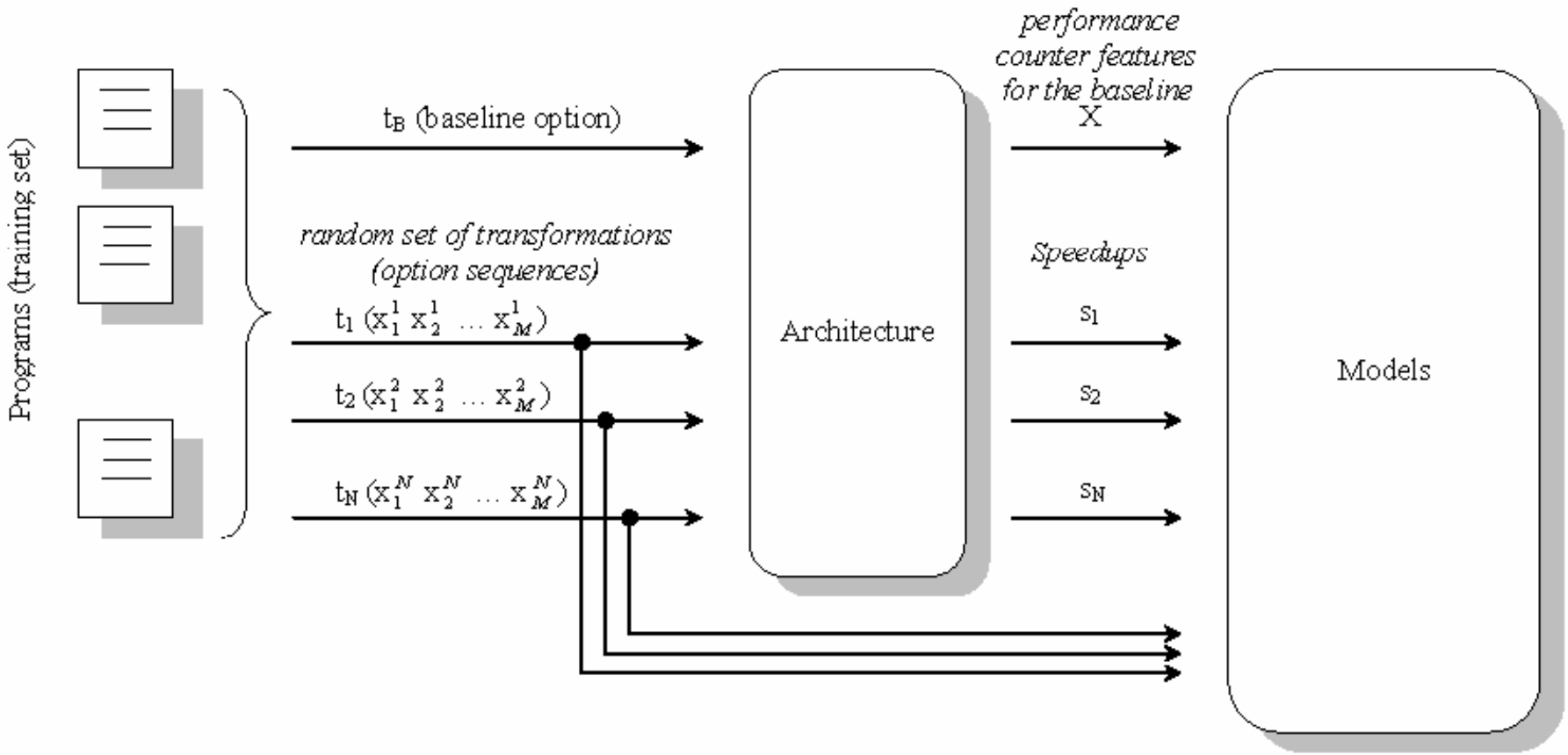
Development websites:

http://gcc-ici.sourceforge.net

http://open64-ici.sourceforge.net

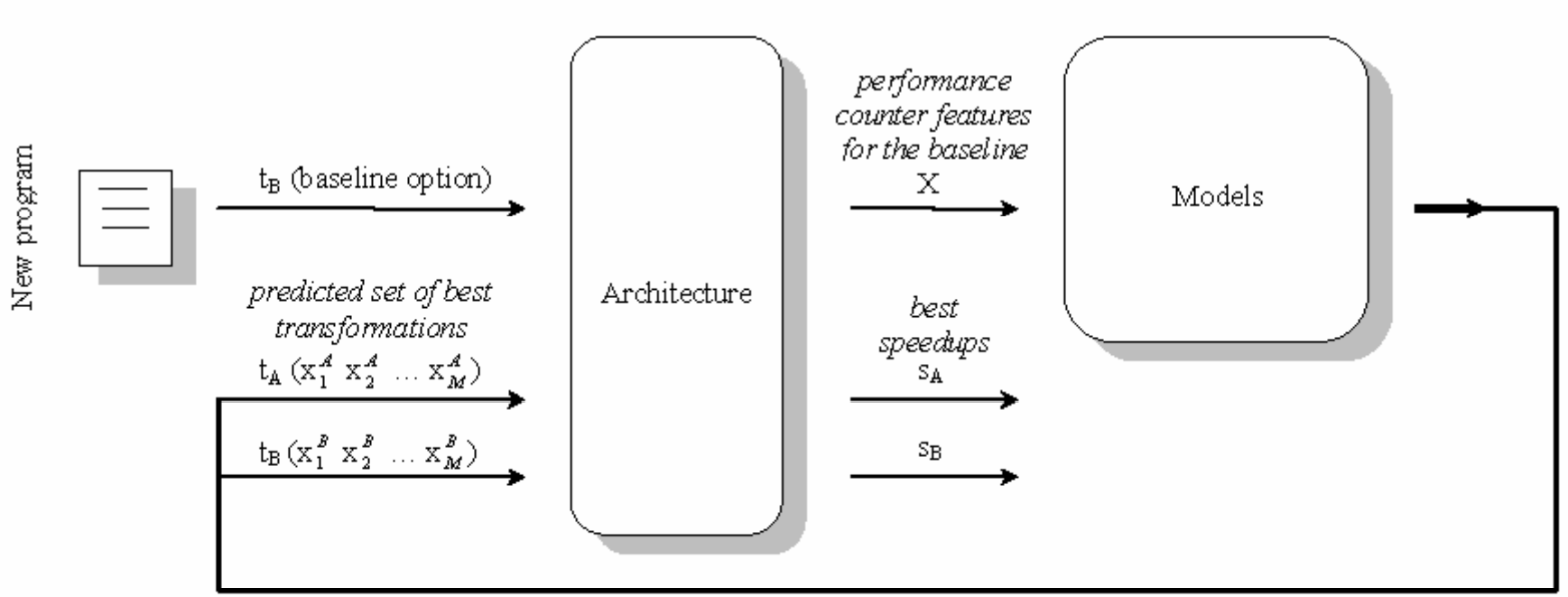
Machine learning

Using predictive modeling and static/dynamic program features to focus search for best optimizations



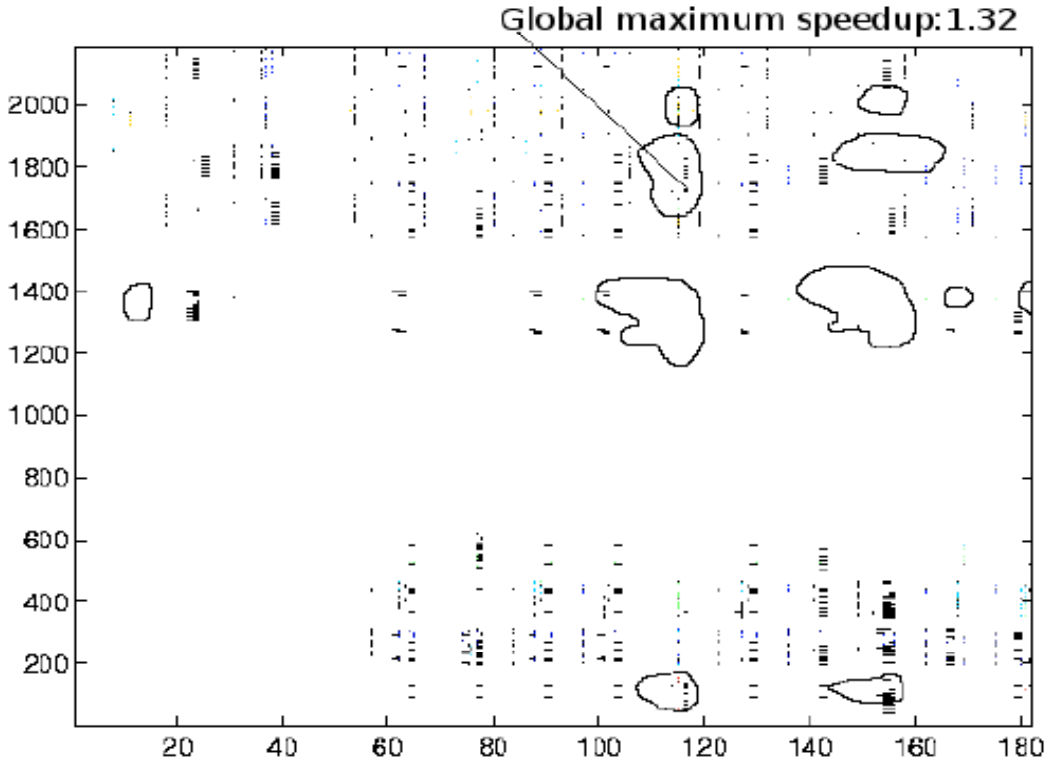
Building / training model

Machine learning



Using model for a new program

Machine learning



Search space = **396000** program transformations

*Predict **2..10** best transformations from this space based on program features and previous optimization experience*

Focusing search (off-line training):

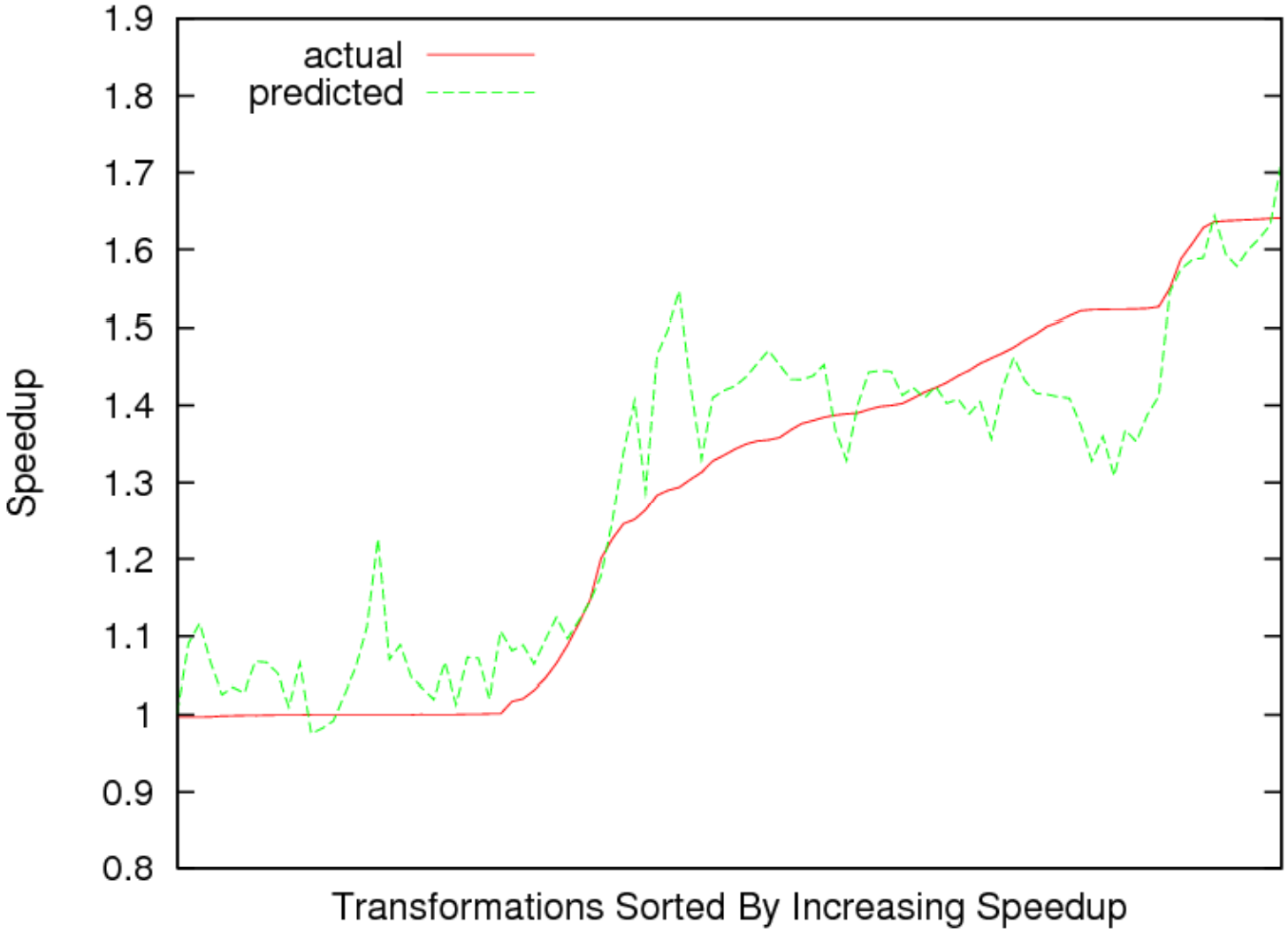
- Independent identically distributed (IID) model
- Markov model

Predicting best transformation for a new program:

- Static features
- Nearest neighbors classifier

Machine learning

Speeding up Architecture Design Space Exploration



Reliable performance model after a few probes → fast search

Machine learning

- F. Agakov, E. Bonilla, J.Cavazos, B.Franke, G. Fursin, M.F.P. O'Boyle, J.Thomson, M. Toussaint and C.K.I. Williams. Using Machine Learning to Focus Iterative Optimization. *Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO)*, New York, NY, USA, March 2006

(best presentation award)

- John Cavazos, Christophe Dubach, Felix Agakov, Edwin Bonilla, Michael F.P. O'Boyle, Grigori Fursin and Olivier Temam. Automatic Performance Model Construction for the Fast Software Exploration of New Hardware Designs. International Conference on Compilers, Architecture, And Synthesis For Embedded Systems (CASES 2006), Seoul, Korea, October 2006

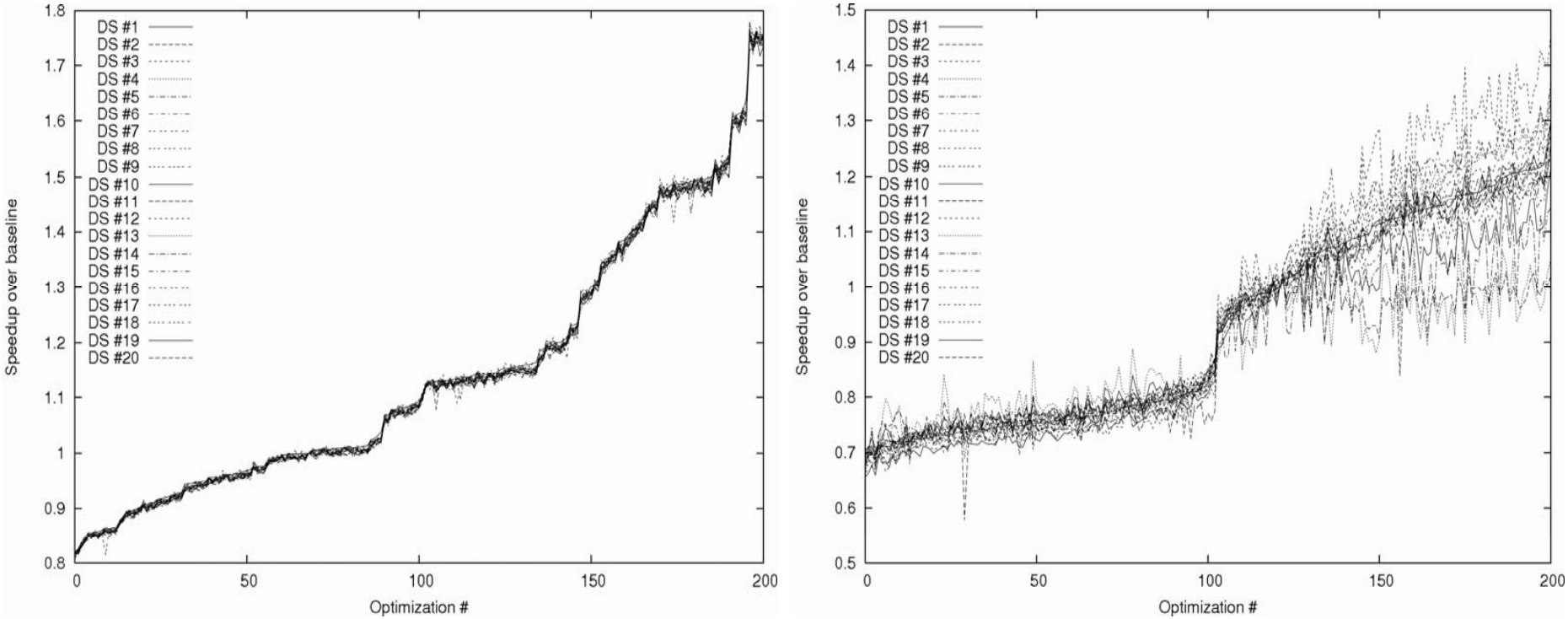
(finalist best paper award)

- John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F.P.O'Boyle and Olivier Temam. Rapidly Selecting Good Compiler Optimizations using Performance Counters. Proceedings of the 5th Annual International Symposium on Code Generation and Optimization (CGO), San Jose, USA, March 2007

- Grigori Fursin and Albert Cohen. Building a Practical Iterative Interactive Compiler. *1st International Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'07),* Ghent, Belgium, January 2007

- Christophe Dubach, John Cavazos, Björn Franke, Grigori Fursin, Michael O'Boyle and Oliver Temam. Enabling fast compiler optimization evaluation via code-features based performance predictor. *ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2007

Dataset sensitivity



- Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization. Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2007), Ghent, Belgium, January 2007

Development website:
http://midatasets.sourceforge.net

Run-time adaptation

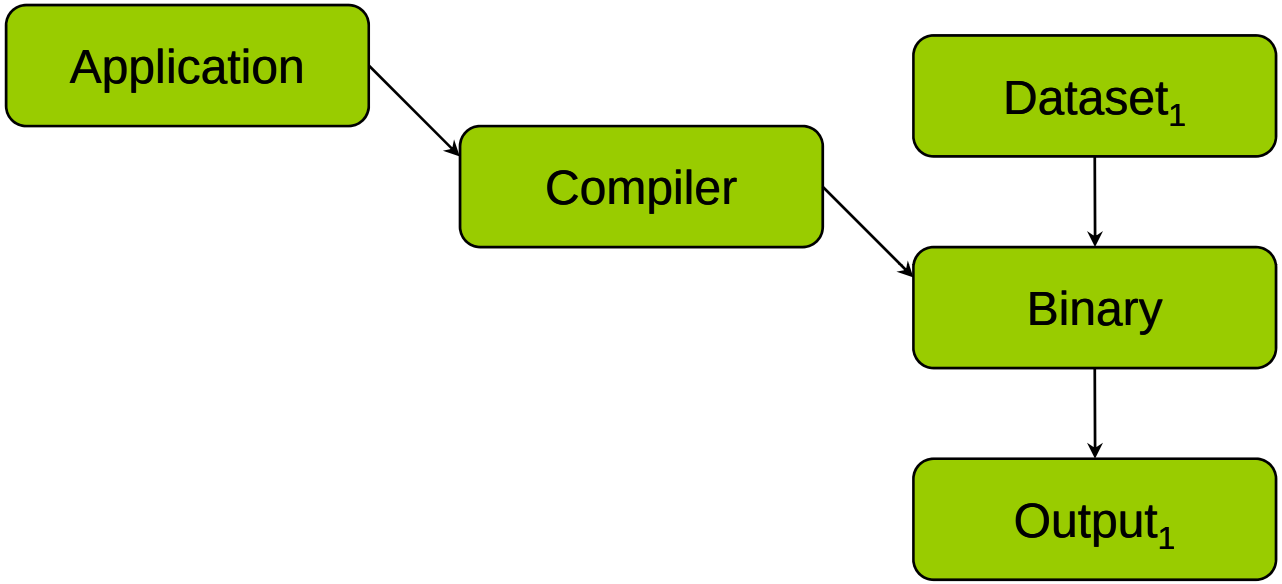
Why adapt at run-time?

- Different program context
- Different run-time behavior
- Different system load
- Different available resources
- Different power consumption
- Different architectures & ISA

For each case we want to find and use best optimization settings!

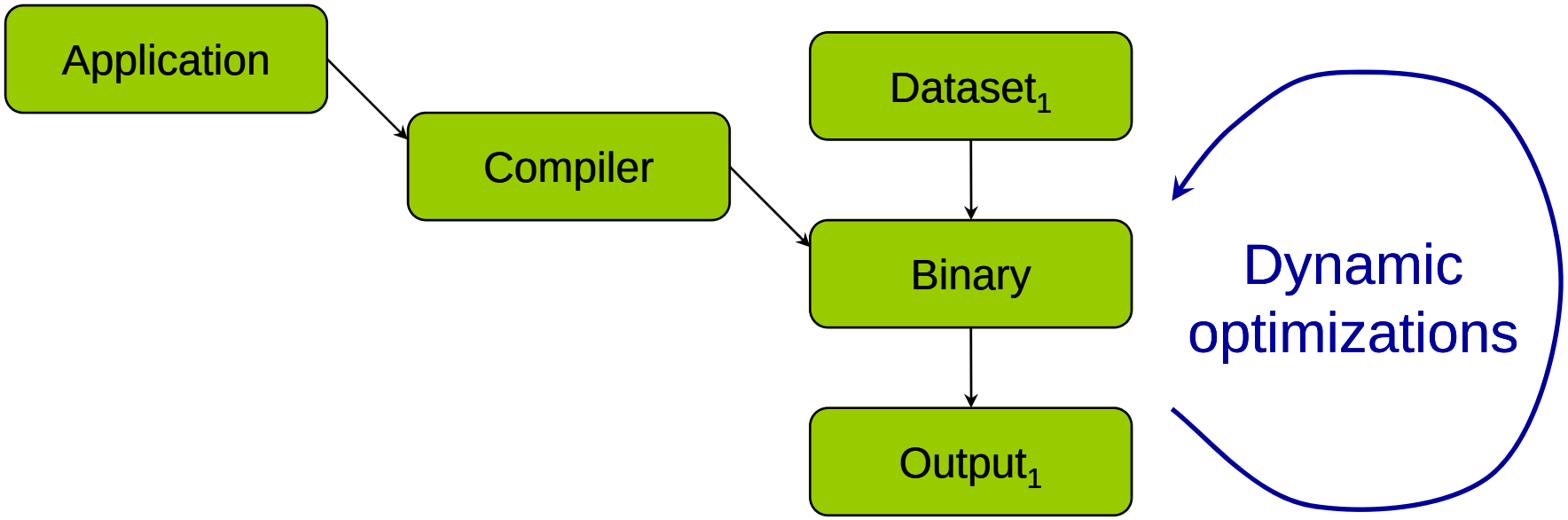
Current methods

Some existing solutions:



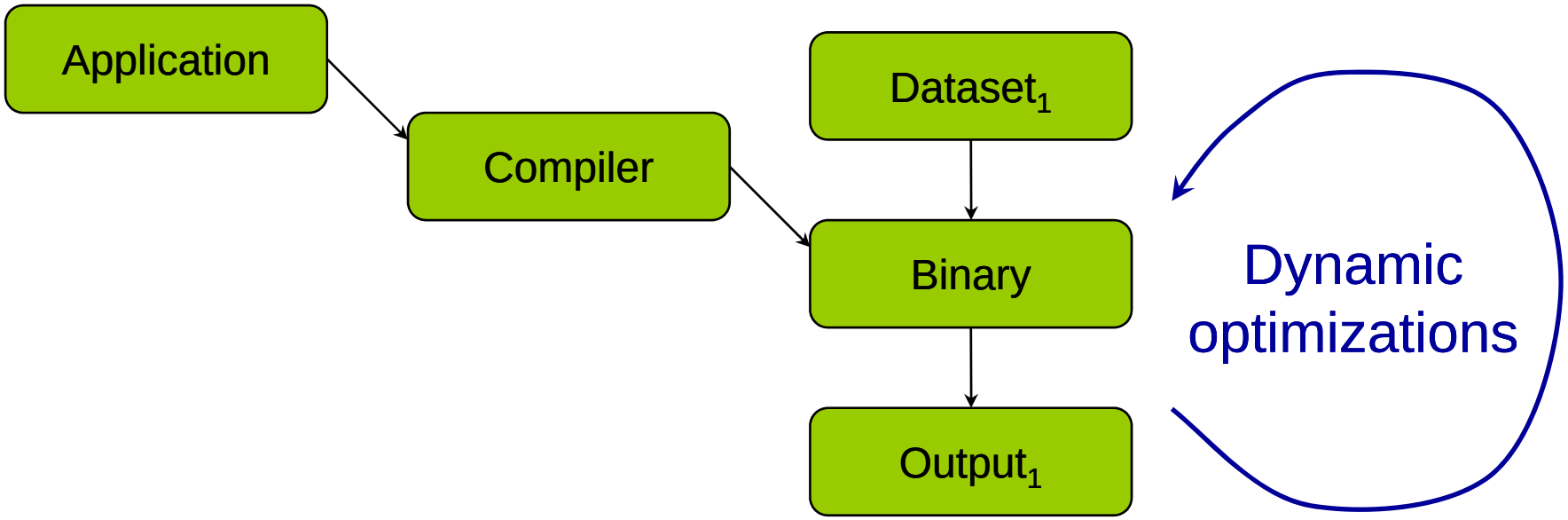
Current methods

Some existing solutions:



Current methods

Some existing solutions:

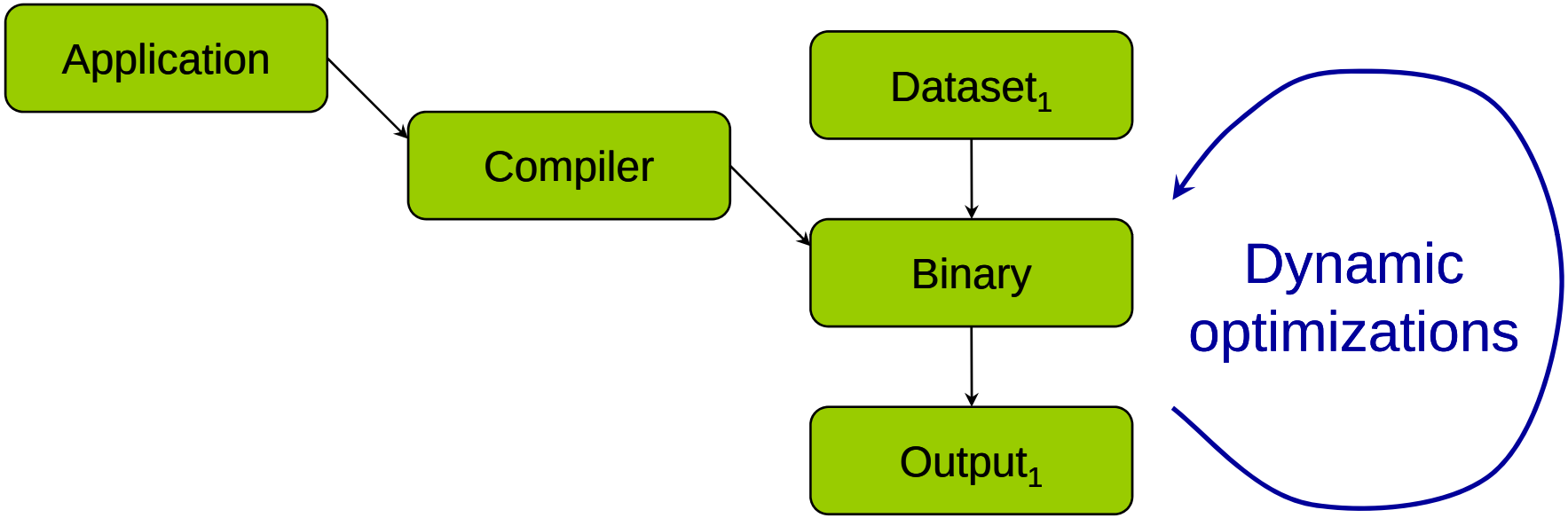


Pros: *run-time information,*

potentially more than one dataset

Current methods

Some existing solutions:



Pros: *run-time information,*

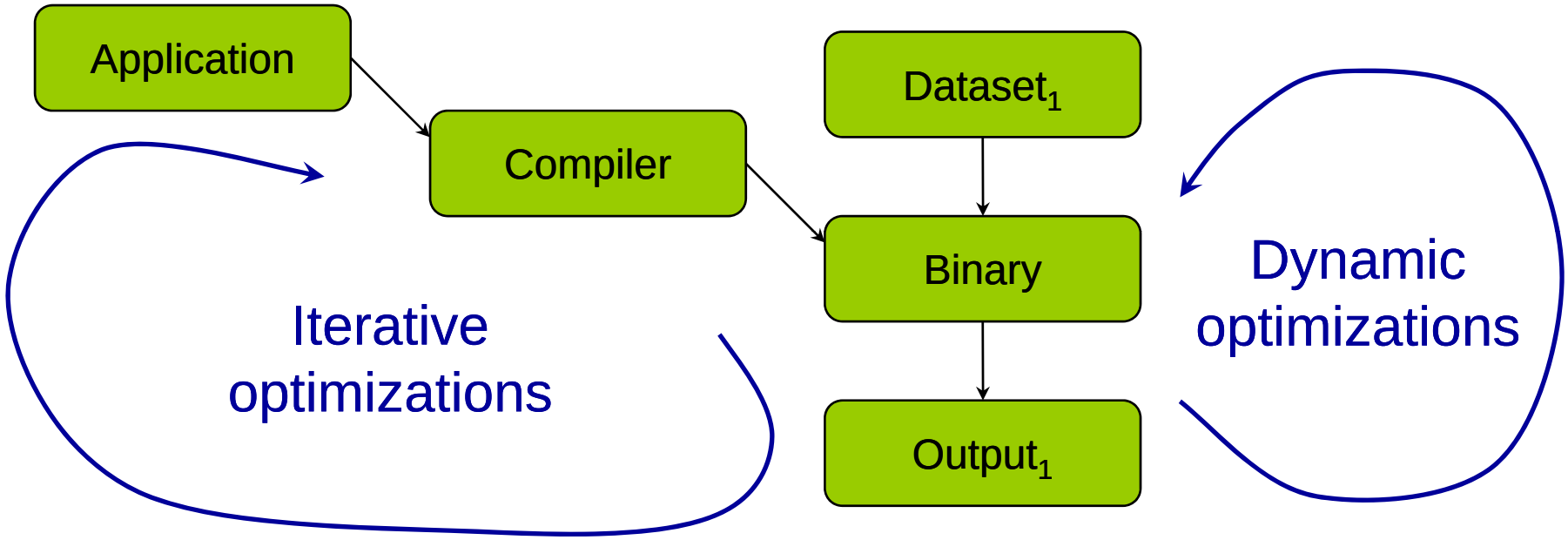
potentially more than one dataset

Cons: *restrictions on optimization time,*

simple optimizations

Current methods

Some existing solutions:



Pros: *run-time information,*

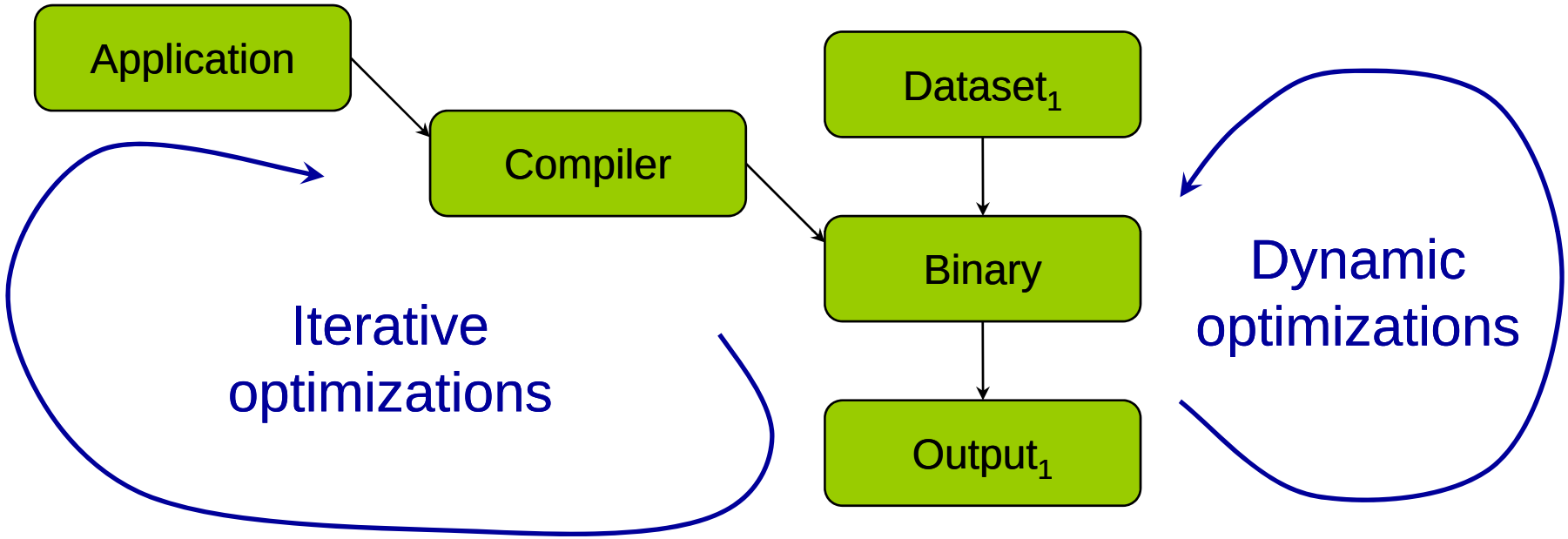
potentially more than one dataset

Cons: *restrictions on optimization time,*

simple optimizations

Current methods

Some existing solutions:



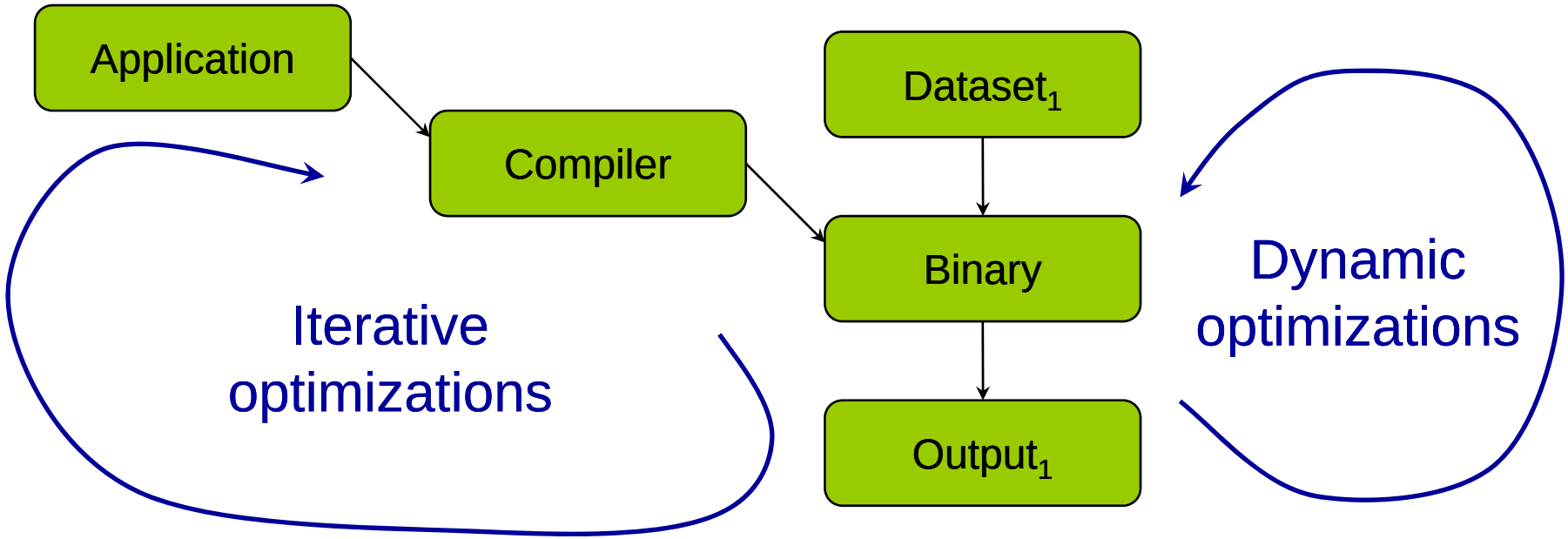
Pros: *powerful transformation space exploration*

Pros: *run-time information, potentially more than one dataset*

Cons: *restrictions on optimization time, simple optimizations*

Current methods

Some existing solutions:



Pros: *powerful transformation space exploration*

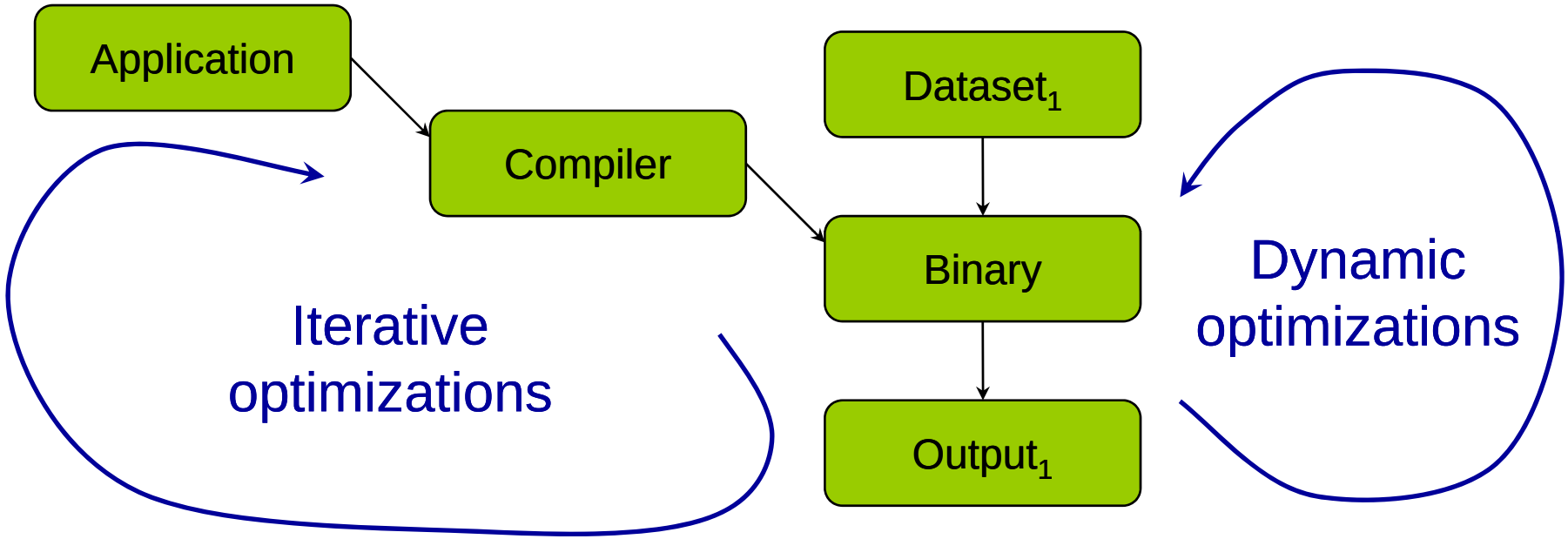
Cons: *slow, one dataset*

Pros: *run-time information, potentially more than one dataset*

Cons: *restrictions on optimization time, simple optimizations*

Current methods

Can we combine both?



Combination of

powerful transformation space exploration,
run-time information
self-adaptable code

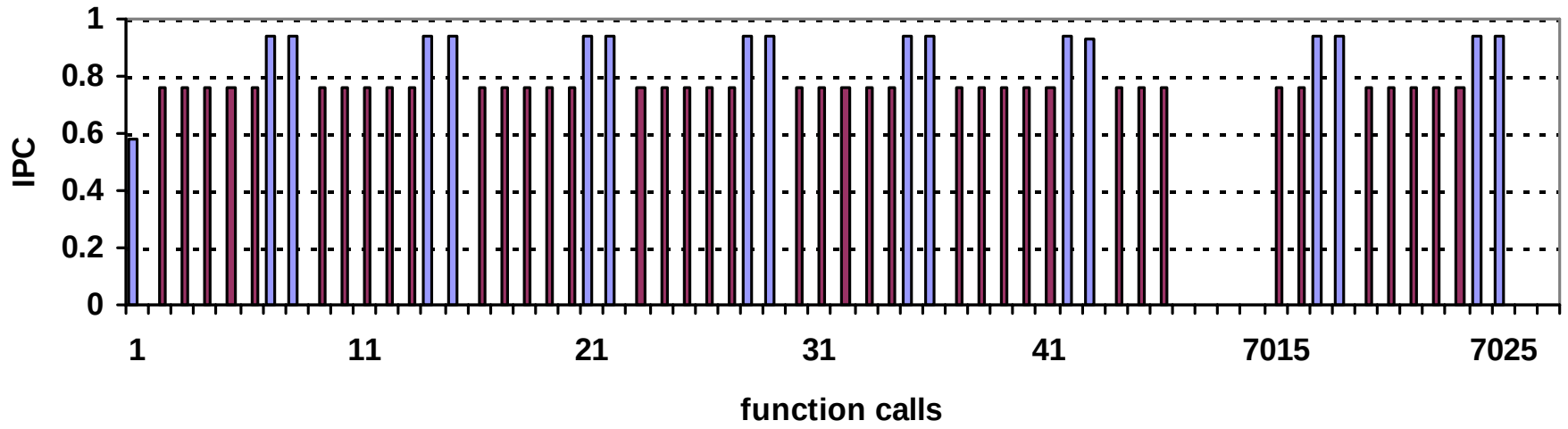
Run-time program adaptation

At which granularity?

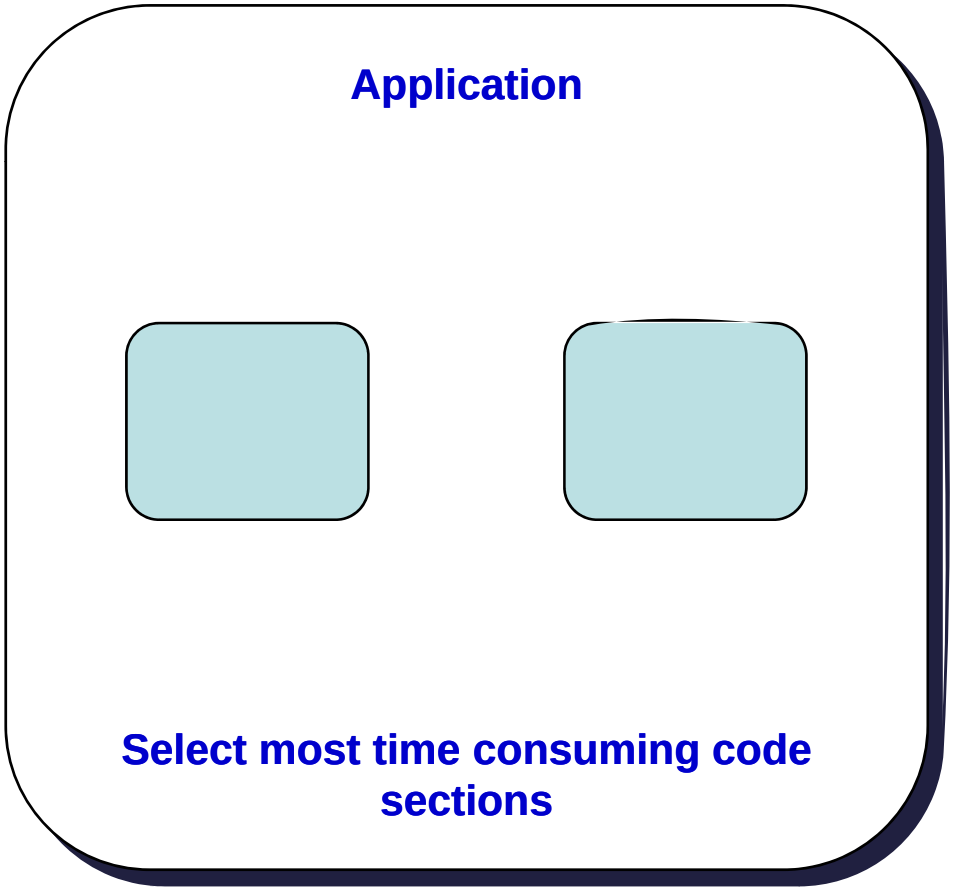
Repeatedly executed time-consuming parts of the code that allow powerful transformations:

typically functions or loops

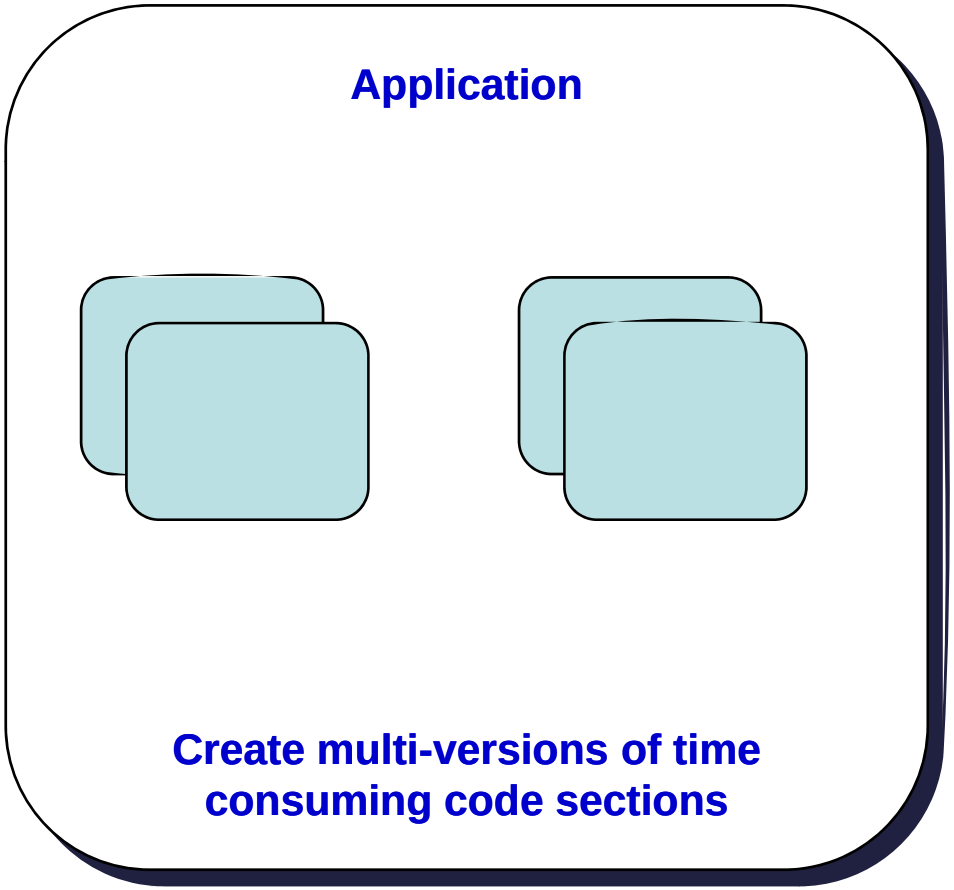
IPC for subroutine resid of benchmark mgrid across calls



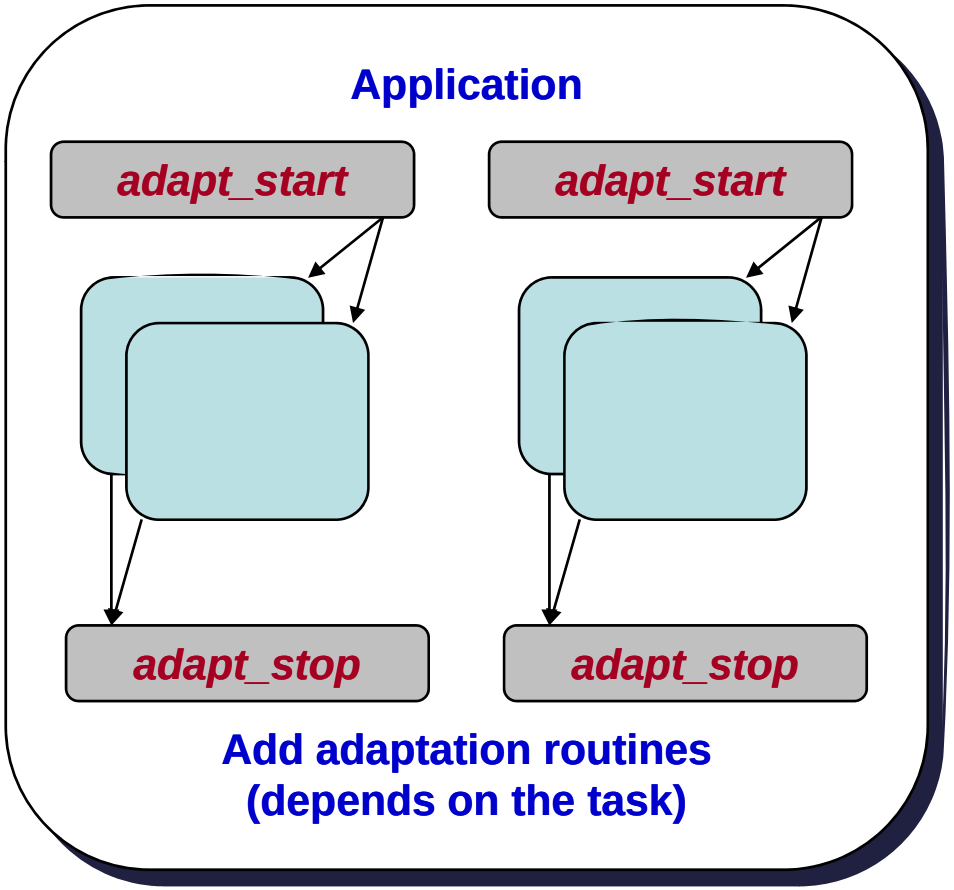
Our approach: static multiversioning



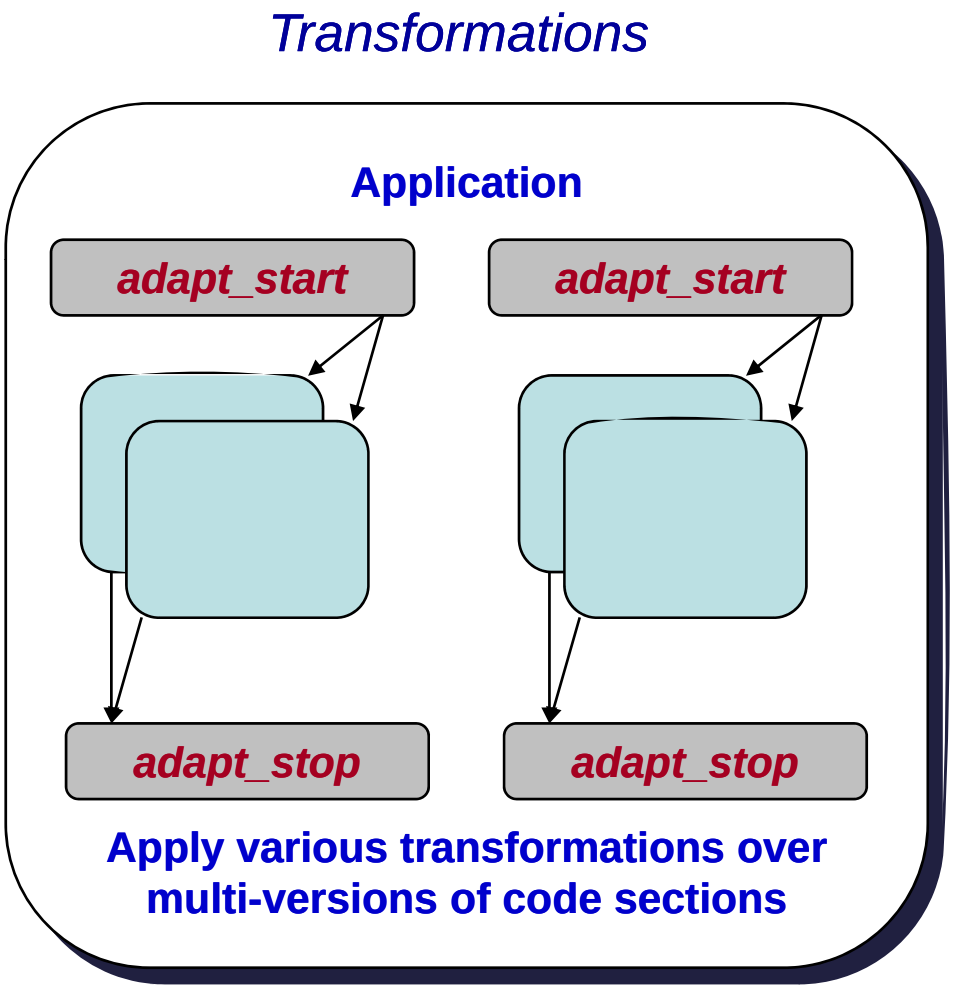
Our approach: static multiversioning



Our approach: static multiversioning

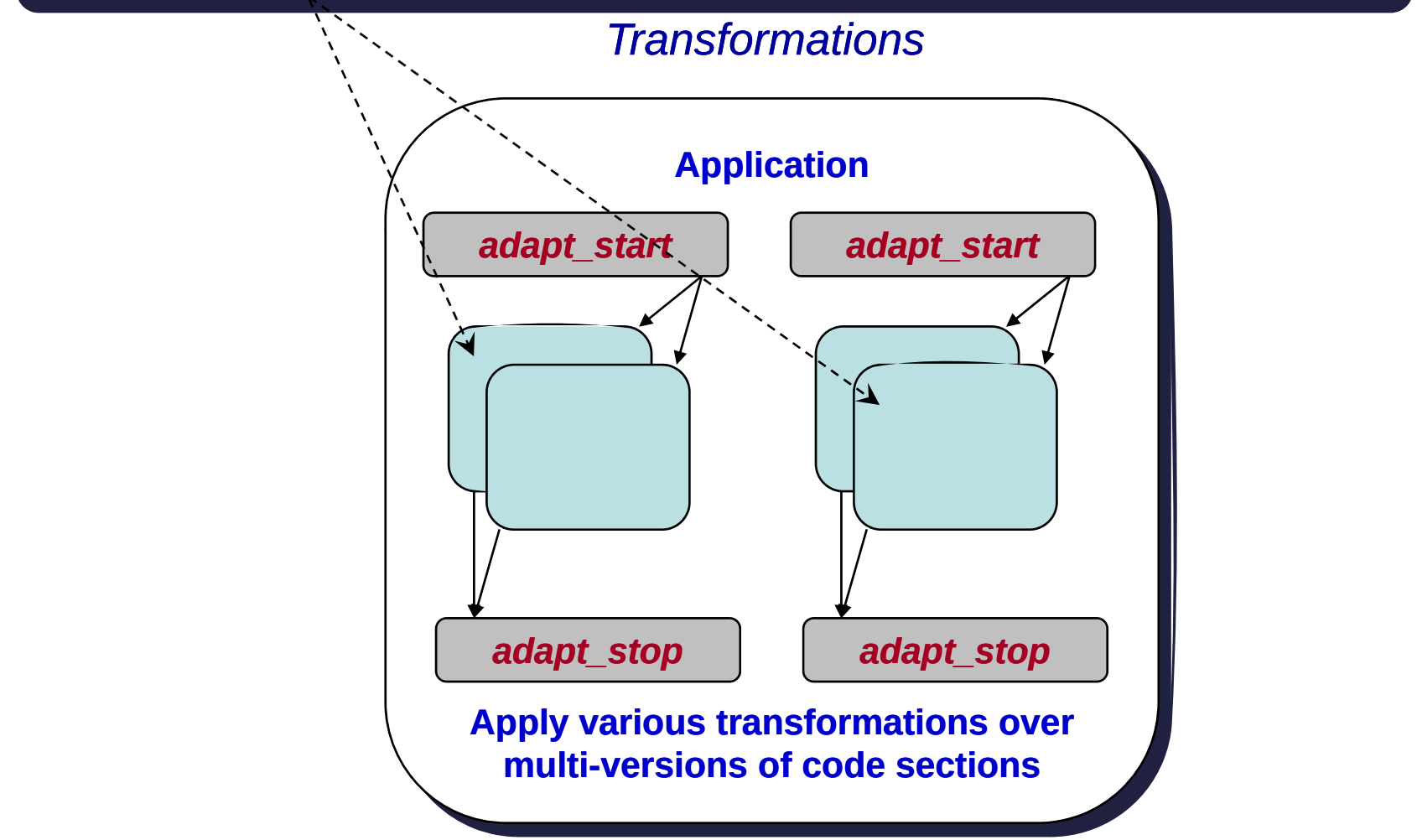


Our approach: static multiversioning

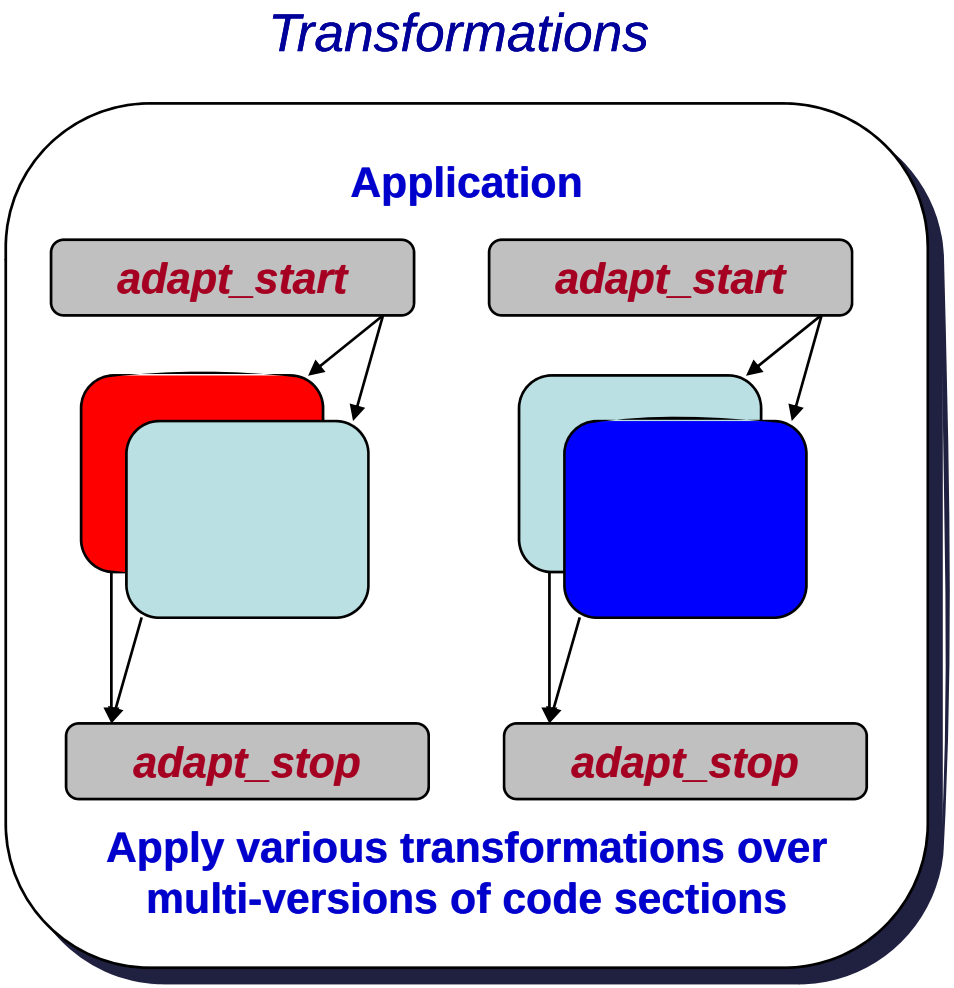


Our approach: static multiversioning

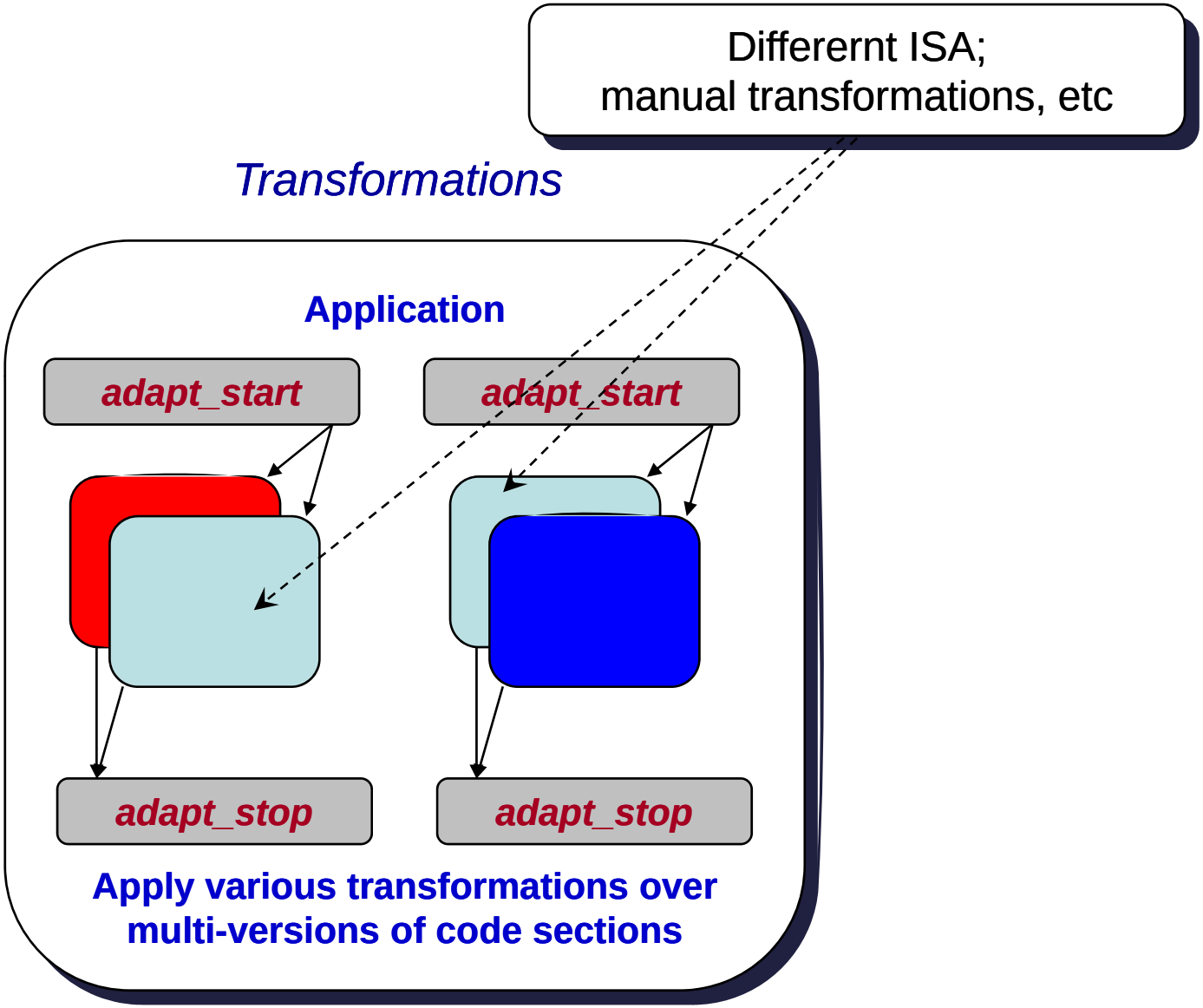
Select global or fine-grain internal compiler optimizations
using Interactive Compilation Interface (ICI)



Our approach: static multiversioning

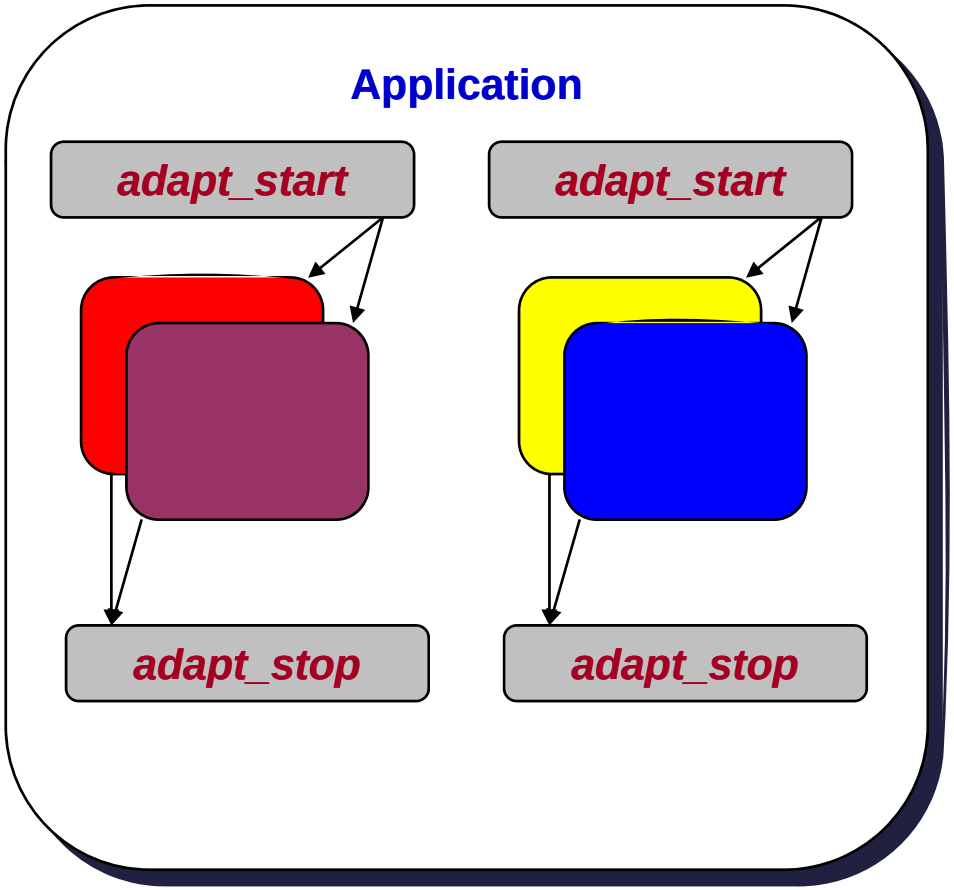


Our approach: static multiversioning



Our approach: static multiversioning

Final instrumented program



What can we do with this framework

- Adapt at run-time for different constraints for statically compiled programs with regular behavior

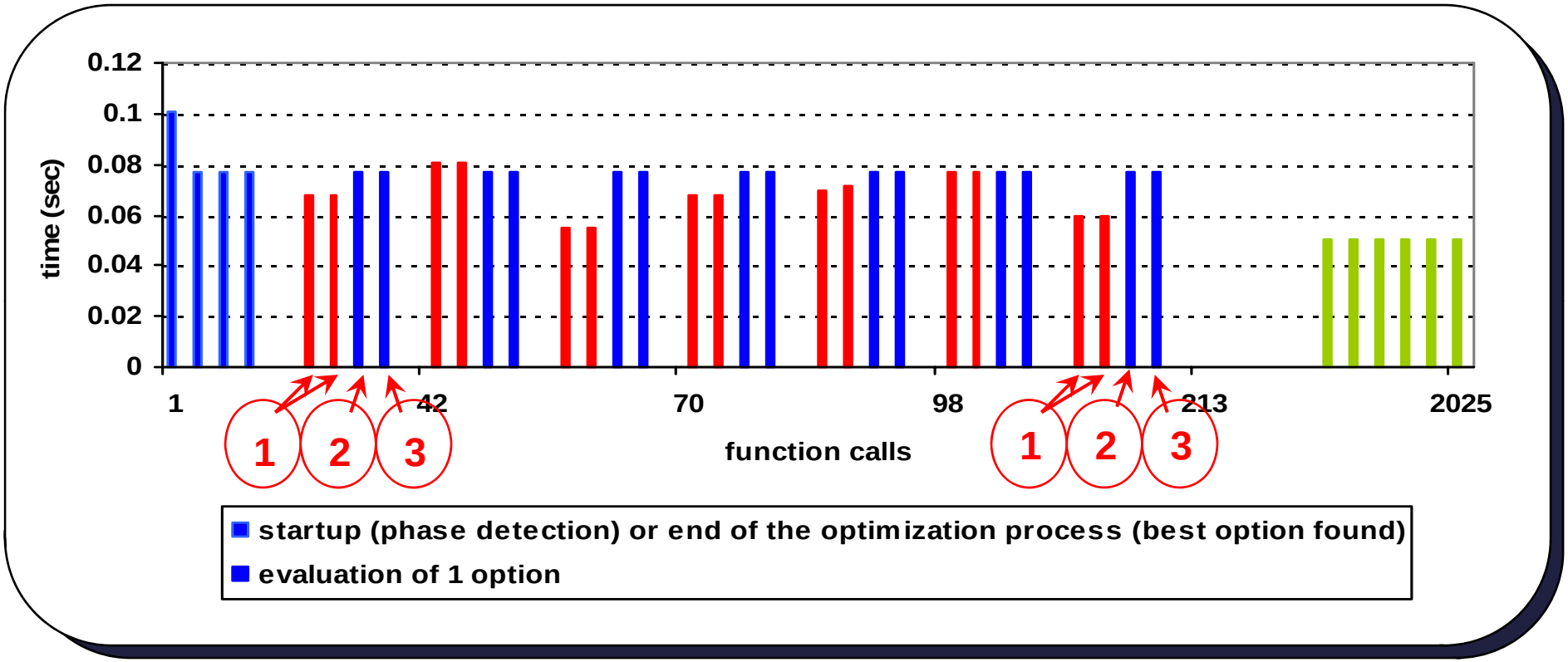
- Determine the effect of optimizations at run-time for programs with varying datasets (realistic continuous collective compilation) by randomly selecting versions at run-time

- Adapt at run-time for different constraints for statically compiled programs with irregular behavior

- Generate mixed multiple-ISA code with run-time adaptation for heterogeneous systems (CPU/GPU or CELL architectures)

Run-time program adaptation

Execution times for subroutine resid of benchmark mgrid across calls



1) Consider new optimization option evaluated after 2 consecutive executions of the code section with the same performance

2) Ignore one next execution to avoid transitional effects

3) Check baseline performance (to verify stability prediction)

Determine the effect of optimizations

Use gprof to collect time spent in functions and clones

$$avt\ (average\ time) = \frac{time\ spent\ in\ function}{number\ of\ calls}, \qquad s\ (speedup) = \frac{avt_{original}}{avt_{cloned}}$$

Continuous Optimization Framework

sequence of evaluations: speedups s_1, s_2, … s_n

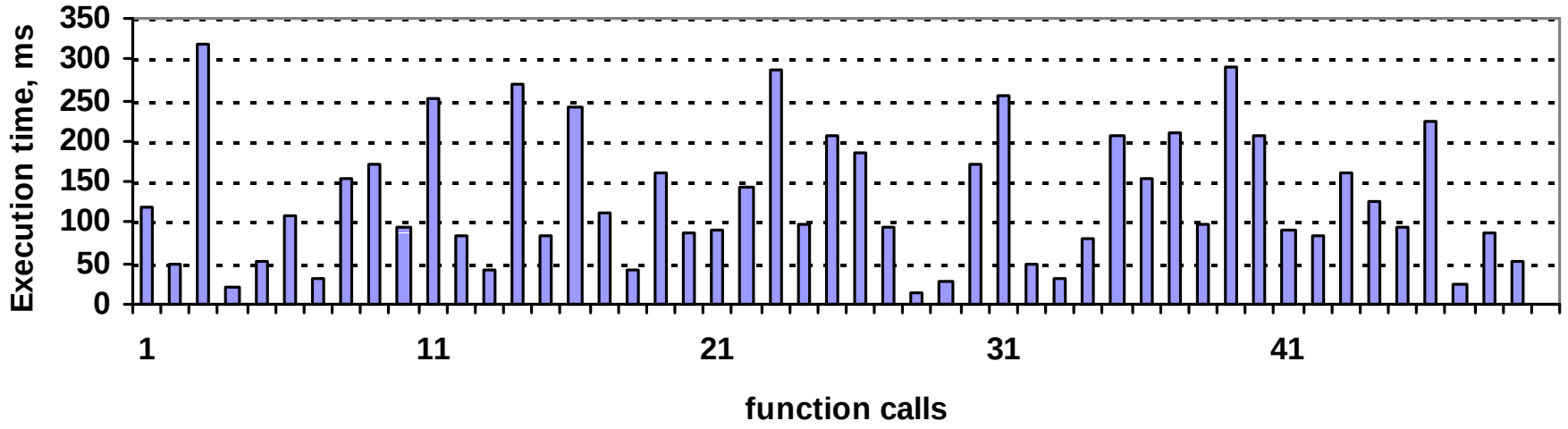
$$e\ (expected\ speedup) = \sum_{i=1}^{n} s_i / n$$

$$v\ (variance) = \sum_{i=1}^{n} (s_i - e)^2$$

Continuously monitor the variance to detect convergence across executions

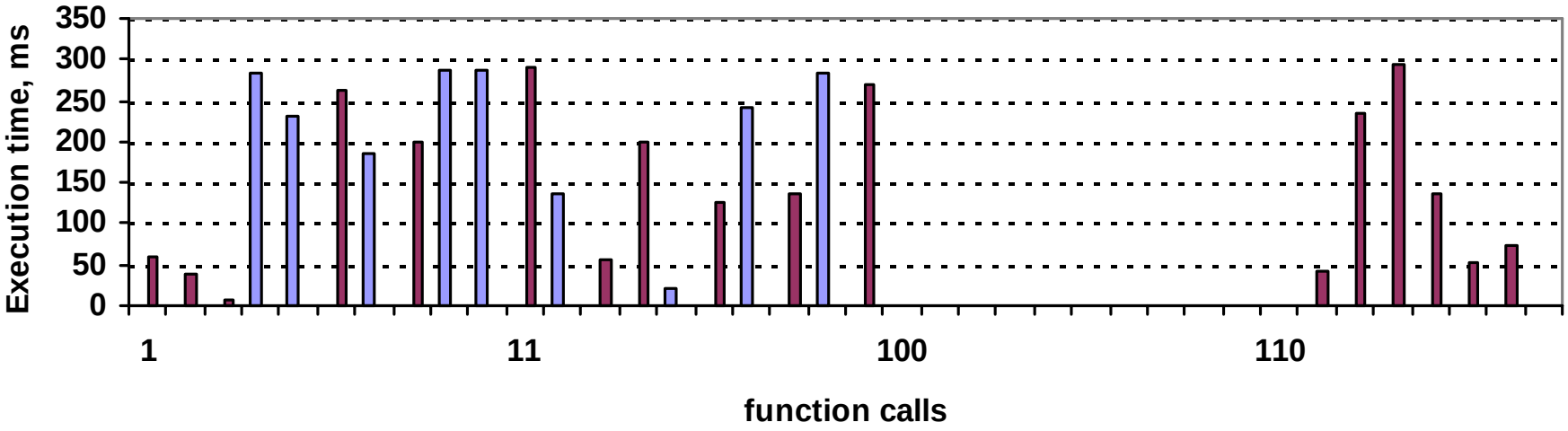
Adaptation for irregular behaviour

Execution time for library subroutine matmul (with 2 different versions)



Adaptation for irregular behaviour

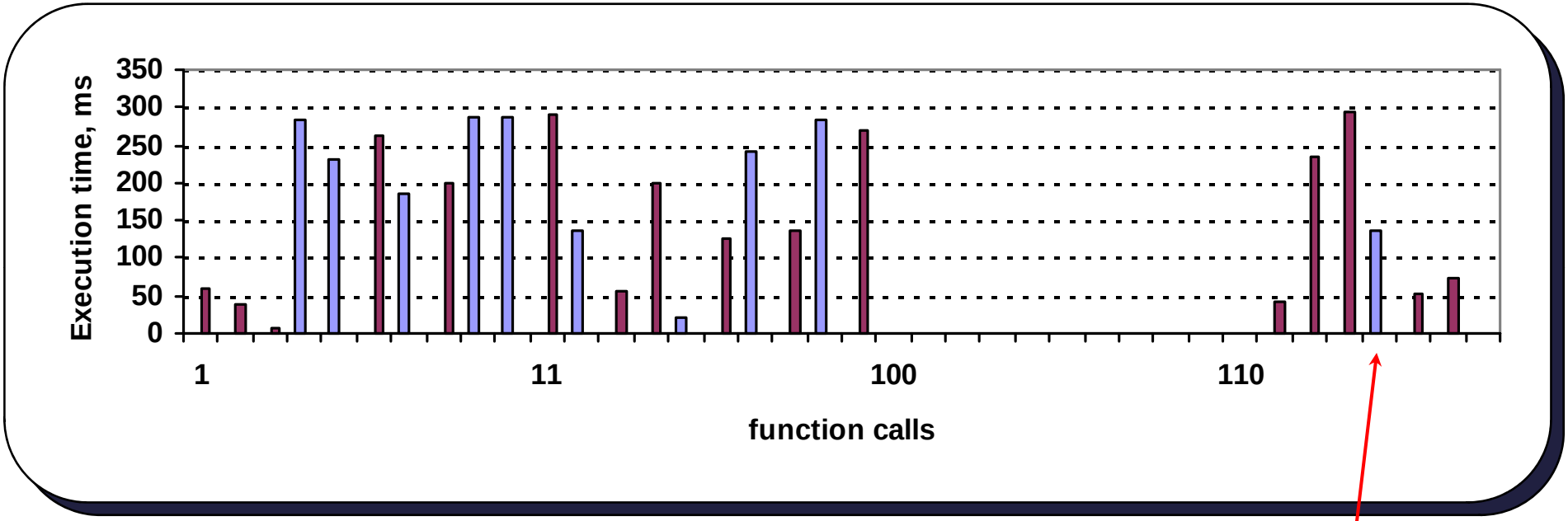
Execution time for library subroutine matmul (with 2 different versions)



- Select versions randomly during a time slot
- At each step calculate execution time per function call and variance
- When variance for all versions is less than some threshold select the best one

Adaptation for irregular behaviour

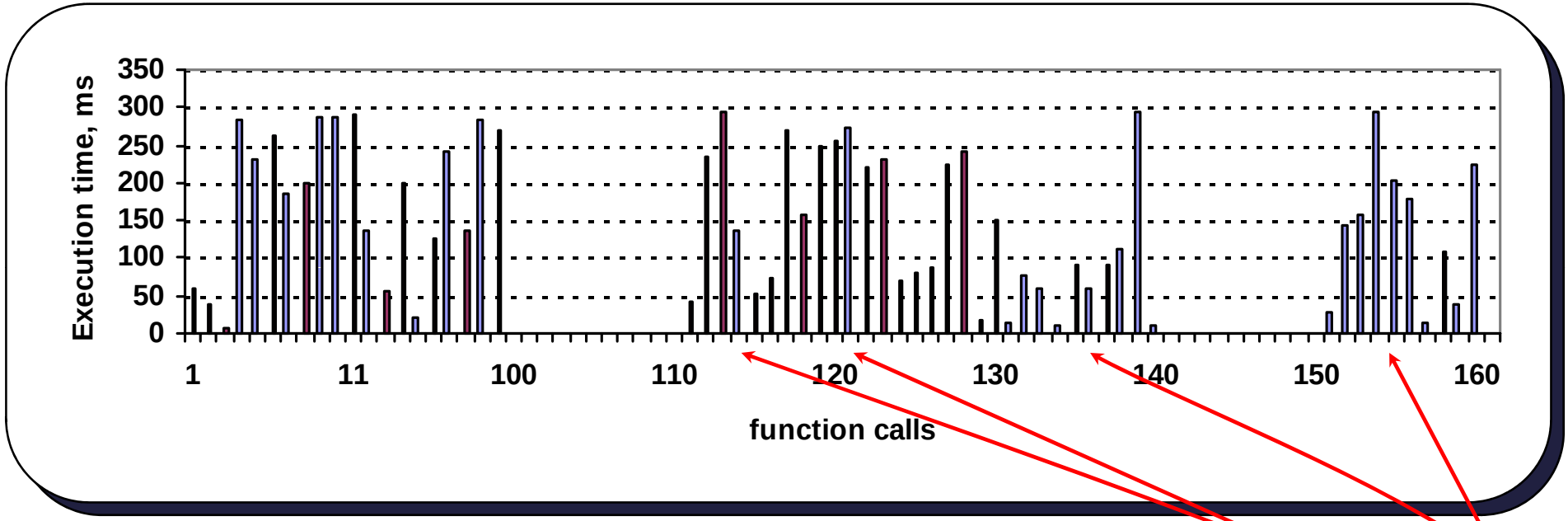
Execution time for library subroutine matmul (with 2 different versions)



- Select versions randomly during a time slot
- At each step calculate execution time per function call and variance
- When variance for all versions is less than some threshold select the best one
- Periodically select non-best version to check if behavior changed

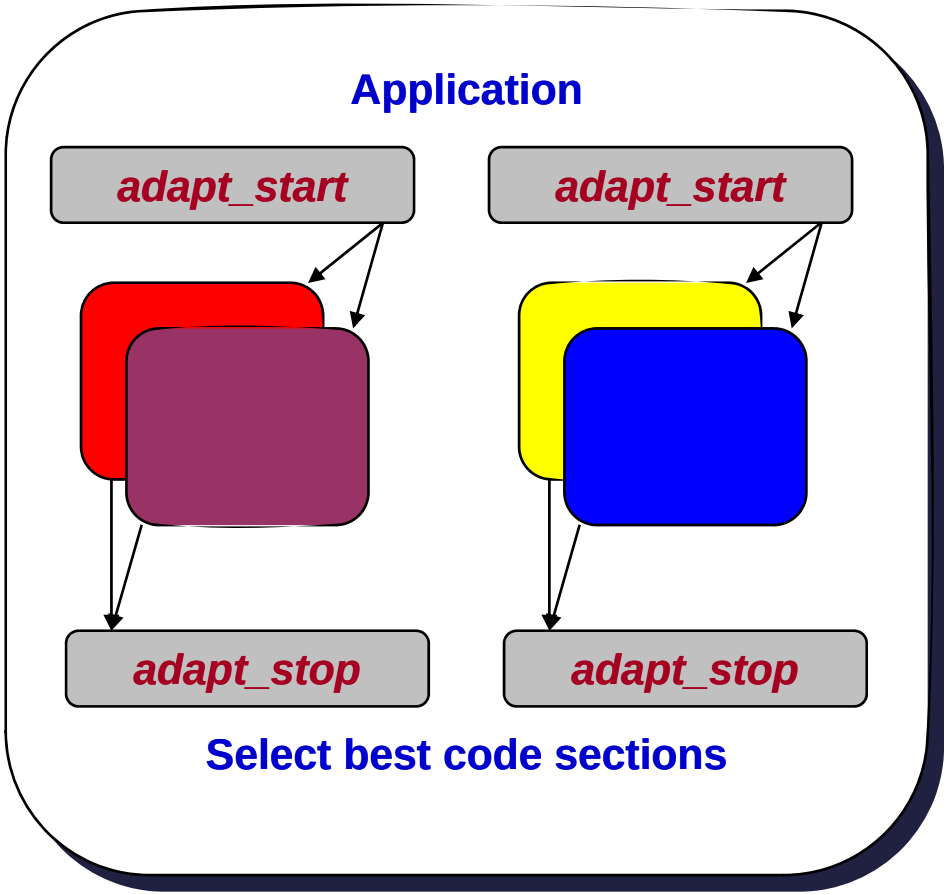
Adaptation for irregular behaviour

Execution time for library subroutine matmul (with 2 different versions)



- Select versions randomly during a time slot (adaptation slot)
- At each step calculate execution time per function call and variance
- When variance for all versions is less than some threshold select the best one
- Periodically select non-best version to check if behavior changed
- If the variance increases, adapt again

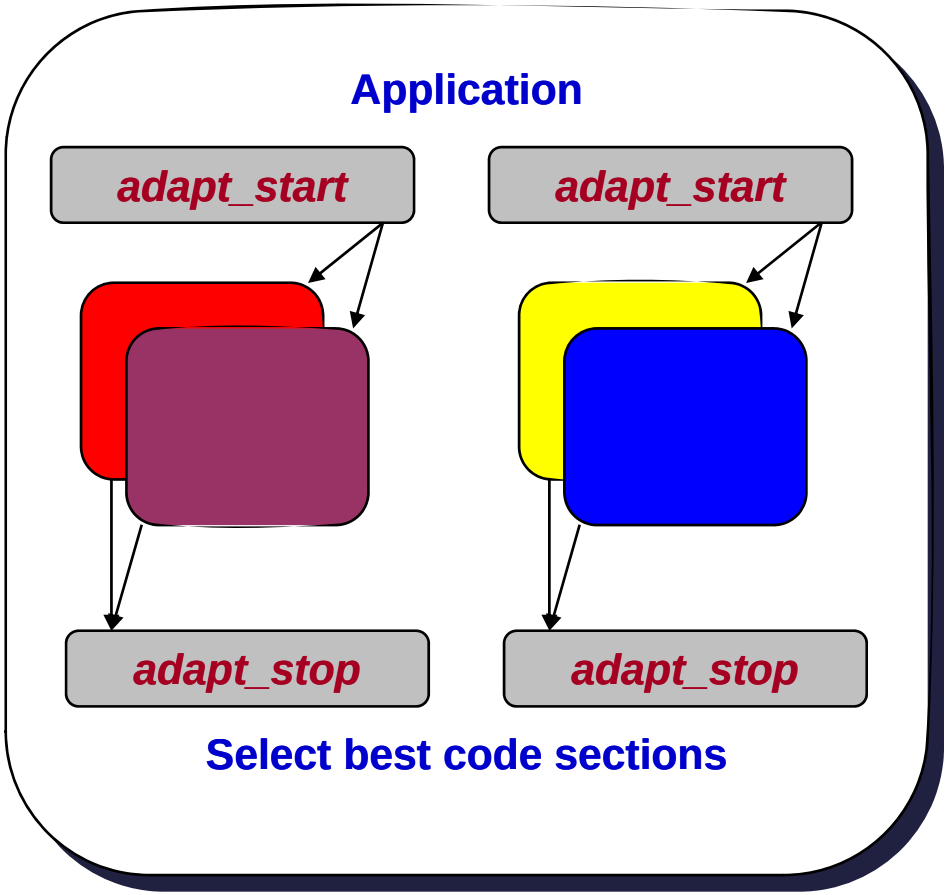
Removing adaptation overhead



Calls to adaptation routines are not direct but through array of functions:

```
static void (*call1[ .. ])();
static void (*call2[ .. ])();
```

Removing adaptation overhead

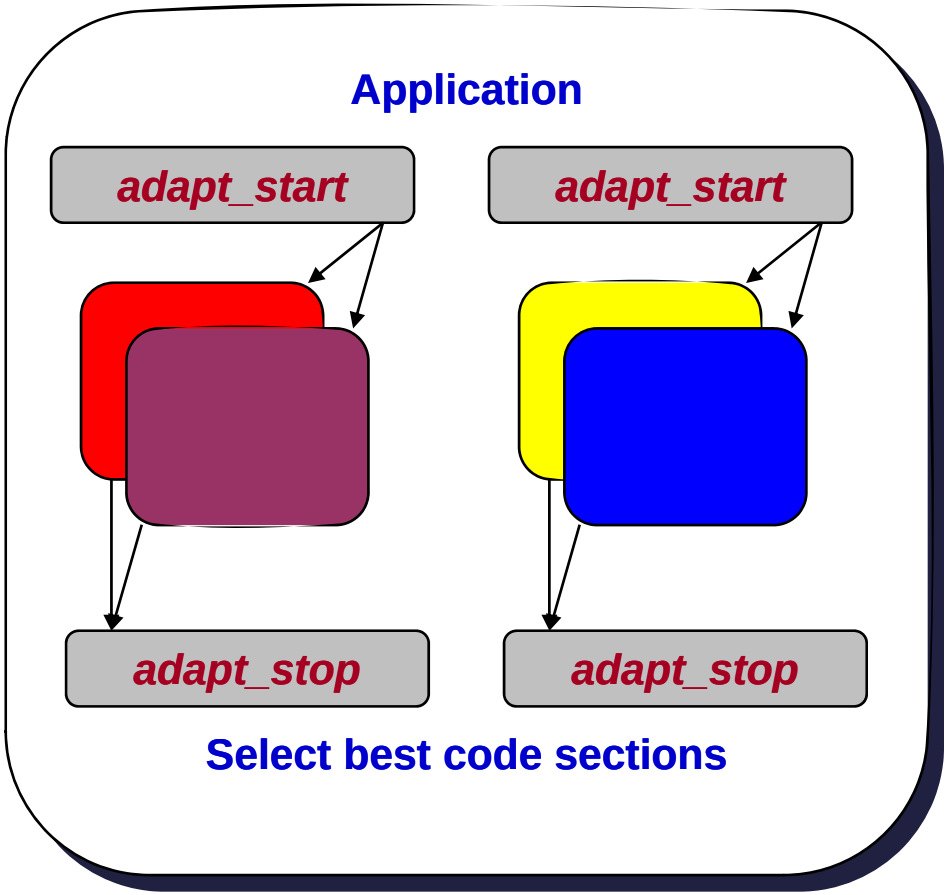


Calls to adaptation routines are not direct but through array of functions:

```
static void (*call1[ .. ])();
static void (*call2[ .. ])();
```

If high-overhead is detected – substitute call with dummy function

Removing adaptation overhead



Calls to adaptation routines are not direct but through array of functions:

static void (*call1[..])();
static void (*call2[..])();

If high-overhead is detected – substitute call with dummy function

To be able to adapt to new program behaviour later at run-time, periodically restore all calls to adptation routines

Run-time program adaptation

- Grigori Fursin, Albert Cohen, Michael O'Boyle and Olivier Temam. A Practical Method For Quickly Evaluating Program Optimizations. *Proceedings of the 1st International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005)*, number 3793 in LNCS, pages 29-46, Barcelona, Spain, November 2005

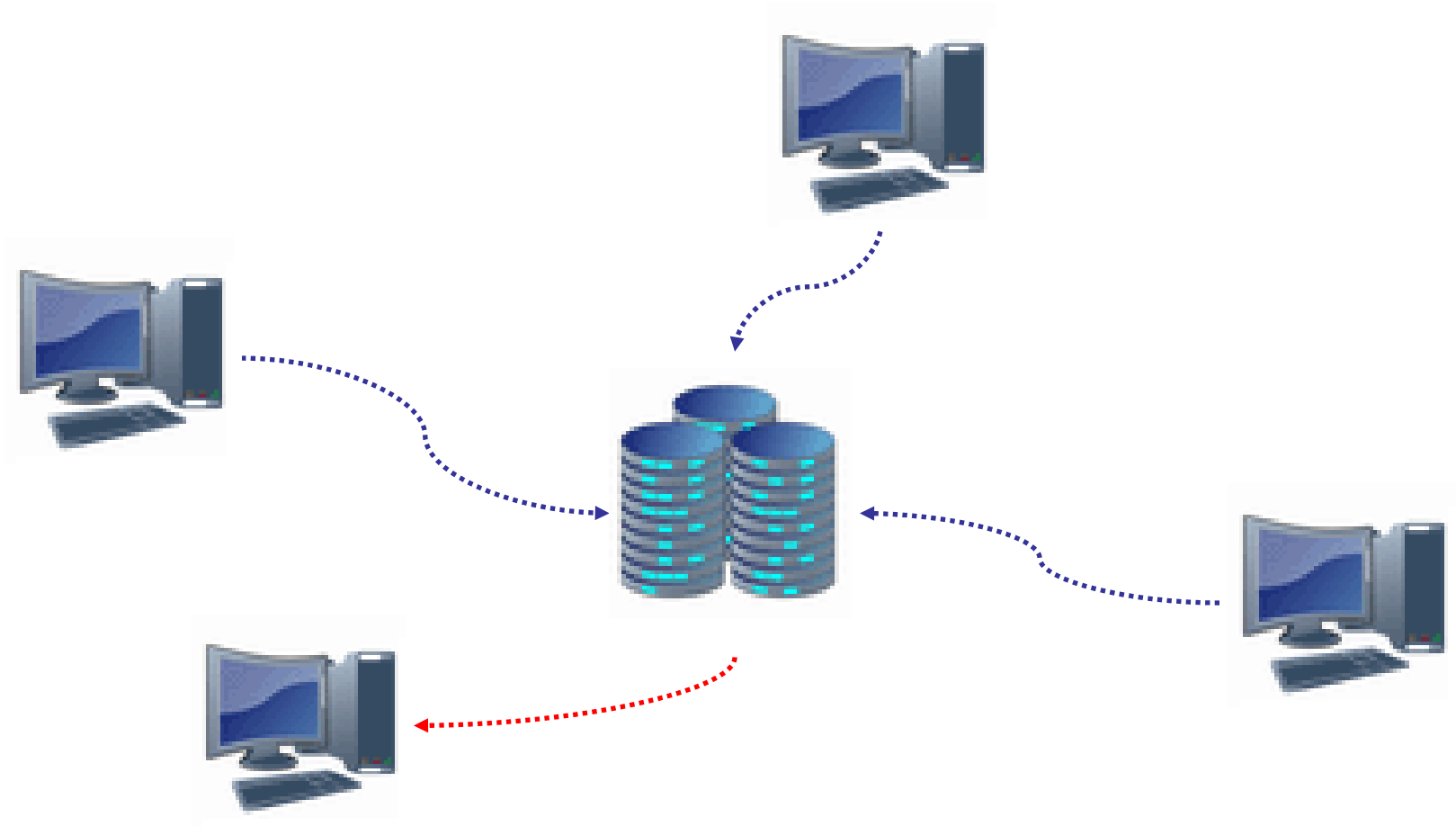
(highest ranked paper, 18% acceptance rate)

Integration of the run-time adaptation into mainline GCC:

- Grigori Fursin, Cupertino Miranda, Sebastian Pop, Albert Cohen and Olivier Temam. Practical run-time adaptation with procedure cloning to enable continuous collective compilation. *GCC Developers' Summit*. Ottawa, Canada, July 2007.

HiPEAC cluster funding to "Explore optimization techniques and runtime code selection mechanisms for heterogeneous systems" for 18 months starting from September, 2006. Collaboration with STMicro, IBM Haifa, UPC

Continuous Collective Compilation



Continuously and transparently collect all optimization knowledge across programs, architectures, datasets, transformations spaces, etc and enable automatic run-time adaptation for different constraints

Software developments

Run-time adaptation:

- Use framework to detect the effect of optimizations at run-time for programs with varying datasets to continuously tune compiler optimization heuristic
- Use the framework for run-time adaptation for statically compiled programs (finish implementing low-overhead adaptation technique in GCC)
- Generate mixed multiple-ISA code for heterogeneous systems (CPU/GPU or CELL)

http://unidapt.sourceforge.net

Interactive Compilation Interface:

- Extend Interactive Compilation Interface to optimize programs at fine-grain level (and tune cost models) with external tools (and machine learning) & move toward modular compiler with transformation plug-ins

http://gcc-ici.sourceforge.net

http://open64-ici.sourceforge.net

Software developments

Continuous Collective Compilation:

• Release 1st version of Continuous Collective Compilation framework (September, 2007)

http://gcc-ccc.sourceforge.net

Machine Learning:

• Improve Machine Learning techniques to automatically tune compiler optimization heuristic and ease future compiler developments

http://www.milepost.eu

Collaborations

Academia:

- INRIA Rennes, INRIA TAO
- Edinburgh University, UK
- UPC, Spain
- UIUC, USA

Industry:

- IBM Research Lab, Israel
- ST Micro, Switzerland
- ARC, United Kingdom
- ARM, United Kingdom
- NXP, former Philips Semiconductors, Netherlands

Projects:

- HiPEAC
- Milepost
- SARC

Thank You !

Machine Learning for Embedded Programs Optimisation *(MILEPOST)*

http://www.milepost.eu

Network of Excellence on High Performance Embedded Architectures and Compilers *(HiPEAC)*

http://www.hipeac.net

Contact email: *grigori.fursin@inria.fr*

More information about research projects and software:
http://fursin.net/research