Autotuning in High-Performance Computing Applications

PRASANNA BALAPRAKASH, JACK DONGARRA , TODD GAMBLIN, MARY HALL, JEFFREY K. HOLLINGSWORTH, BOYANA NORRIS, AND RICHARD VUDUC

PROCEEDINGS OF THE IEEE | Vol. 106, No. 11, November 2018

What is Autotuning

- Automatic generation of a search space of possible implementations of a computation that are evaluated through models and/or empirical measurement to identify the most desirable implementation
- For HPC applications, based on the architecture, automatically generate executables that give good performance.

Why Autotuning

- Profound differences in architecture and programming models.
- A desirable feature of high-performance applications is performance portability, whereby the same application code can achieve high performance across a diversity of architectures.
- Manually rewriting code for a new supercomputer architecture or architecture generation is prohibitively expensive and limits the porting of applications to new platforms.

Types of Autotuning

- Libraries
- Compilers & Code Generators
- Application-Level Autotuning
- Frameworks and Domain-Specific Systems

Libraries

- Optimize the performance-critical subroutines.
- Examples: ATLAS
  - Generates efficient code by running a series of timing experiments to determine optimal code structures.
- Pros
  - Applications naturally rely on libraries
  - Common APIs are widely used, so it is as simple as linking to a different implementation.
  - Eliminate the need for programmer involvement in autotuning
- Cons
  - Libraries are limited in the scope of their applicability
  - Hard for optimizations beyond individual library calls without contextual information

Compilers & Code Generators

- Special compilers that do auto-tuning based on source code
- Examples: CHiLL, Orio, POET.
- The compiler needs a search space. But it can be fixed for common algorithms and can be guided by experts
- Study shows compiler-directed autotuning can produce code that achieves performance comparable to and sometimes exceeding that of manual tuning

Application-Level Autotuning

- High-level tuning that may change the algorithms or program structures.
- Pros:
  - Significant algorithmic changes can be expressed.
  - It allows the use of auto-tuning options that can change the output of the program
  - Programmers can express code variants along with meta information that aids the system in variant section at runtime
- Cons
  - each application developer must specify the autotuning

Frameworks and Domain-Specific Systems

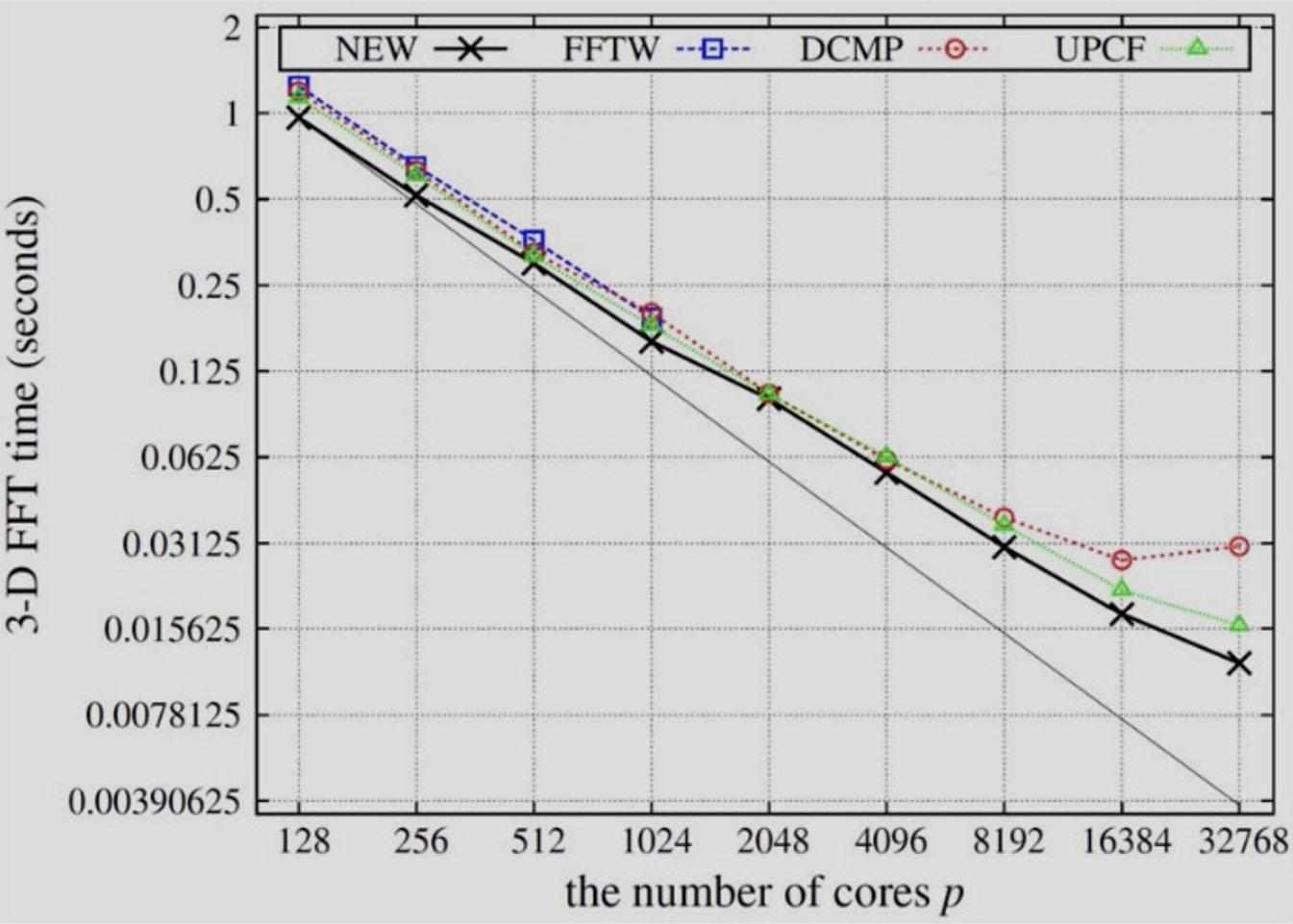
- High-level frameworks that provide template abstractions that users can use to guide their code.
- Examples: Apollo, RAJA, Kokkos.
- Advantages:
  - Clearly separates tuning concerns from application semantics: application developers write the code, and performance experts can put high-level framework code.
  - Allows code variant selection to be implemented as an external library, so that decision models can be updated over time

Search Method

- All last three autotuning requires to search in a space to find the optimal.
- To improve the scalability of large search space, machine learning algorithms are used.
- Global algorithms such as simulated annealing, genetic algorithms.
- Local algorithms such as Nelder-Mead simplex, orthogonal search.
- Model-based selection algorithms to avoid actually running code.

Case Studies

- Using autotuning compiler CHiLL on LOBPCG, it only needs 7 lines of input code whereas manually tuning needs to change 2000 lines of code. It also outperforms manually tuned code by 3%.
- Autotuned FFT library shows better strong scaling performance.
- RAJA framework achieves 2.5x speedup on Sedov problems and 15% speedup on ARES (hotspot problem)


NEW
FFTW
DCMP
UPCF
3-D FFT time (seconds)
2
1
0.5
0.25
0.125
0.0625
0.03125
0.015625
0.0078125
0.00390625
128
256
512
1024
2048
4096
8192
16384
32768
the number of cores p

Thanks!