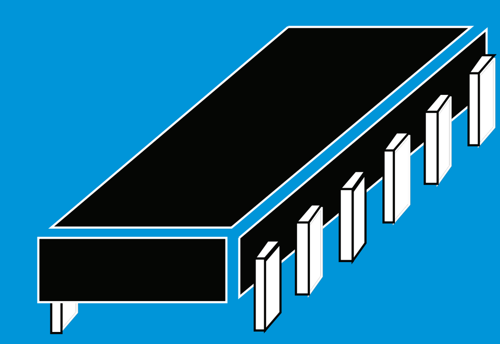
Center for Computational Sciences, University of Tsukuba
www.ccs.tsukuba.ac.jp

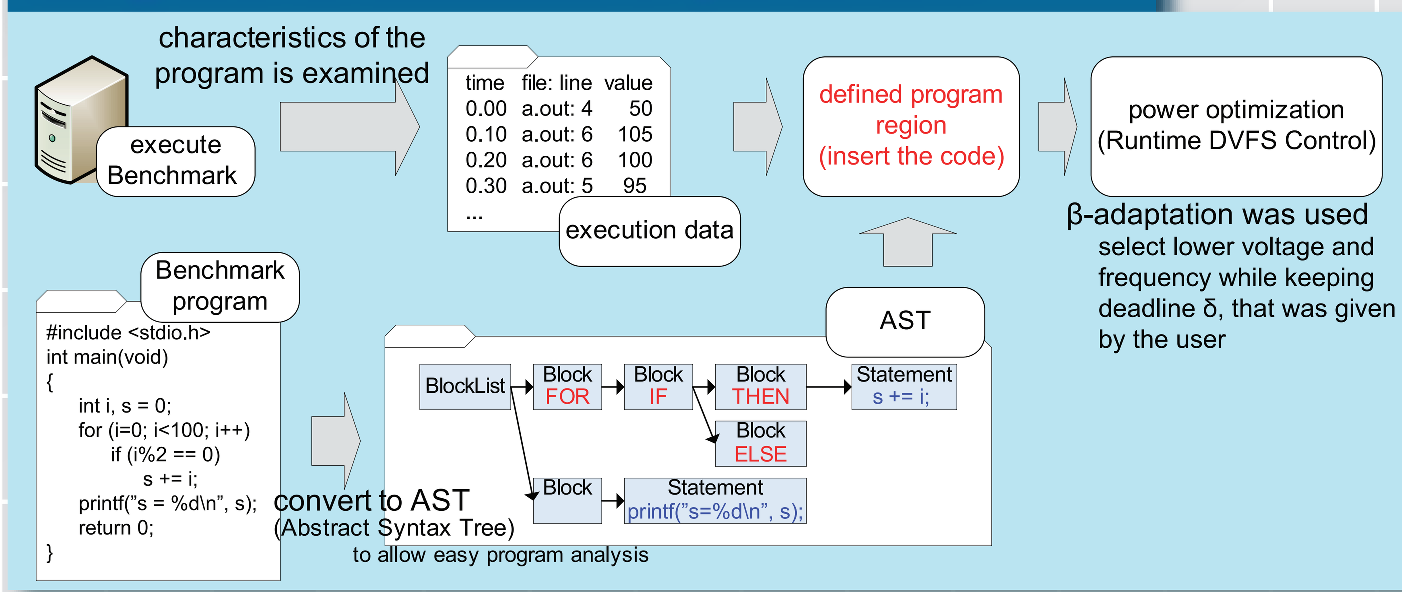


High Performance Computing Research

Runtime DVFS Control with instrumented code

- Code-Instrumented (CI-) Runtime DVFS control
  - manages the voltages and frequencies at the instrumented code at runtime
  - achieves better energy reduction than Interrupt-based (IB-) Runtime DVFS control method
    - ► avoid the fluctuation of performance by program characteristics
  - is easier to use than static (profile-based) DVFS method
    - ► not require data for optimization such as profile
    - ► can use the already proposed Runtime DVFS optimization

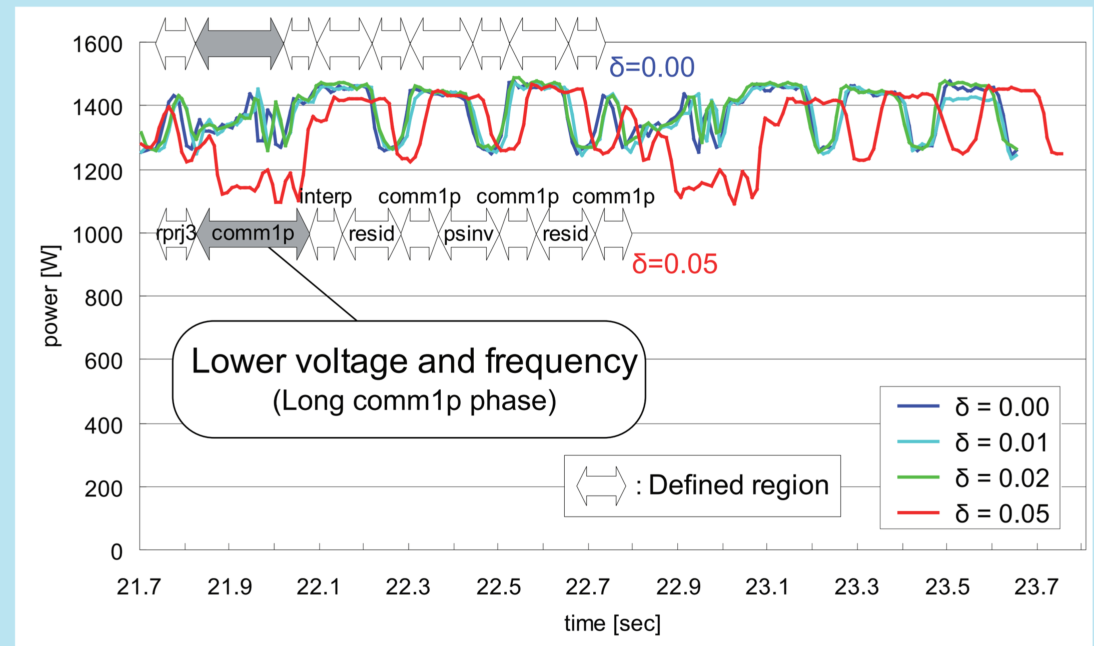
Flowchart of CI-Runtime DVFS method



A comparison between each DVFS method

	IB-Runtime DVFS	CI-Runtime DVFS	Static DVFS (profile-based)
Energy reduction	good (sometimes bad) may not achieve good energy reduction between a calc. and non-calc. phases	better	best may achieve the best energy reduction using profile
User's cost	nothing	defining the region	defining the region, obtaining the profile
Code Instrumented (defining the region)	no (interrupt-based) no pre-execution	yes defining program region ► should have similar characteristics ► should not drastically change the program characteristics	yes
DVFS control	runtime β-adaptation [Chen et.al. 2005] ■ a kind of runtime DVFS control algorithm ■ select the gear using relationship between frequency and actual performance ► use only the performance information ► don't use the power consumption data	runtime	selecting the gears before execution need the profile of each available gear for each region

Result: Power profile (CI-Runtime)



AMD Opteron148, DDR-SDRAM 1GB, GbE, NPB-MG CLASS=C 2 iterations, 16 nodes

- The execution time has increased by 4.49% (keeping deadline δ=0.05)
- Overall energy consumption has increased by 1.21% while δ=0.05. (CPU energy consumption has decreased by 4.01%)
- Overall energy consumption was increased where static power consumption is large

FFTE: A High-Performance FFT Library

- FFTE is a Fortran subroutine library for computing the Fast Fourier Transform (FFT) in one or more dimensions.
- It includes complex, mixed-radix and parallel transforms.
- FFTE is typically faster than other publically-available FFT implementations, and is even competitive with vendor-tuned libraries.

Features

HPC Challenge benchmark

- High speed
  - Supports Intel's SSE2/SSE3 instructions.
- Parallel transforms
  - Shared / Distributed memory parallel computers (OpenMP, MPI and OpenMP + MPI)
- High portability
  - Fortran77 + OpenMP + MPI
  - Intel's intrinsics for SSE2/SSE3 instructions.
- HPC Challenge Benchmark
  - FFTE's 1-D parallel FFT routine has been incorporated into the HPC Challenge (HPCC) benchmark.

Approach

- Many FFT routines work well when data sets fit into a cache.
- When a problem size exceeds the cache size, however, the performance of these FFT routines decreases dramatically.
- Some previously presented six-step FFT algorithms require
  - Two multicolumn FFTs.
  - Three data transpositions.

  The chief bottlenecks in cache-based processors.
- We combine the multicolumn FFTs and transpositions to reduce the number of cache misses.

Design

- Performance
  - One goal for large FFTs is to minimize the number of cache misses.
- Ease of use: routine interfaces
  - Similar to sequential SGI SCSL or Intel MKL routines
- Portability
  - Communication: MPI
  - Computation: Fortran77 + OpenMP

Performance of FFTE 4.0

Data:
N1 x N2 x N3 = 2^24 x P

Machines:
Xeon EM64T 3.0GHz
Gigabit Ethernet
1024 MB DDR2/400