The PARSEC Benchmark Suite Tutorial

- PARSEC 3.0 -

By

Yungang Bao, Christian Bienia, Kai Li
Princeton University
Tutorial Contents

- **Part 1:** Understanding PARSEC
  - Overview
    - History, Impact, What’s New
    - Workloads
    - Research on PARSEC

- **Part 2:** Working with PARSEC
  - The parsecmgmt tool
  - Building & Running workloads
  - Configuration files

- **Part 3:** Roadmap of PARSEC
  - Network Workloads
  - GPU version

- **Part 4:** Concluding Remarks
Part 1

Understanding PARSEC
What is PARSEC?

- Princeton Application Repository for Shared-Memory Computers
- Benchmark Suite for Chip-Multiprocessors
- Started as a cooperation between Intel and Princeton University, many more have contributed since then
- Freely available at:
  

Other Resources:


parsec-users@lists.cs.princeton.edu

- You can use it for your research
Goal: An open-source parallel benchmark suite of emerging applications for evaluating multi-core and multiprocessor systems

Application domains: financial, computer vision, physical modeling, future media, content-based search, deduplication

Current release: PARSEC 2.1 (13 applications)
The first version of PARSEC was created by Intel and Princeton University. We would like PARSEC to be a community project. Many people and institutions have already contributed.
Interest in PARSEC

- 6000+ Downloads
Impact of PARSEC

- Google Scholar Citations: 400+
- Citation in top conferences (~40%)
History of PARSEC

- Jan 2008 PARSEC 1.0
  - 12 workloads
- Feb 2009 PARSEC 2.0
  - One new workload, raytrace
- Aug 2009 PARSEC 2.1
  - Bugfix

- PARSEC 3.0
  - Summer 2011
PARSEC 3.0 is coming soon

- New framework
  - Support network workloads
  - Support citations to encourage contribution
  - Be more convenient to add new workloads

- Much improved workloads
  - blackscholes, bodytrack, canneal, dedup, facesim, ferret, fluidanimate, freqmine, vips,

- SPLASH-2 and SPLASH-2x
  - Existing SPLASH-2 using the same framework
    - Use parsecmgmt to manage, build, and run
  - SPLASH-2x (joint work with Prof. JP Singh)
    - Multiple input sets at different scales
Objectives of PARSEC

- Multithreaded Applications
  Future programs must run on multiprocessors

- Emerging Workloads
  Increasing CPU performance enables new applications

- Diverse
  Multiprocessors are being used for more and more tasks

- State-of-Art Techniques
  Algorithms and programming techniques evolve rapidly

- Support Research
  Our goal is insight, not numbers
There aren’t any two workloads with the same combinations.
Blackscholes Overview

- Prices a portfolio of options with the Black-Scholes PDE
- Computational finance application (Intel)
- Synthetic input based on replication of 1,000 real options
- Coarse-granular parallelism, static load-balancing
- Small working sets, negligible communication

Blackscholes is the simplest of all PARSEC workload
Blackscholes Rationale

- Computers have become key technology for trading

- Derivatives are financial instrument with one of highest analytical requirements

- Blackscholes formula fundamental description of option behavior

- High demand for performance: Saving few milliseconds can earn lots of money
Blackscholes Characteristics

Working Sets

- Options
- Portfolio

Cache Hits

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Small working sets, negligible communication
Bodytrack Overview

- Tracks a markerless human body
- Computer vision application (Intel)
- Input is video feed from 4 cameras
- Medium-granular parallelism, dynamic load-balancing
- Pipeline and asynchronous I/O
- Medium working sets, some communication

Output of Bodytrack (Frame 1)
Bodytrack Rationale

- Machines increasingly rely on computer vision to interact with environment
- Often no aid available (e.g. Markers, constrained behavior)
- Must usually happen in real-time

Bodytrack Characteristics

Working Sets

- Edge maps
- Input frames

Cache Hits

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Medium working sets, some communication
Canneal Overview

- Minimizes the routing cost of a chip design with cache-aware simulated annealing
- Electronic Design Automation (EDA) kernel (Princeton)
- Input is a synthetic netlist
- Fine-grain parallelism, no problem decomposition
- Uses atomic instructions to synchronize
- Synchronization strategy based on data race recovery rather than avoidance
- Huge working sets, communication intensity only constrained by cache capacity.

Workload with most demanding memory behavior
Canneal Rationale

- Optimization is one of the most common types of problems.
- Place & Route is a difficult EDA challenge.
- Transistor counts continue to increase at an exponential rate.
- Simulated annealing allows to scale optimization cost by allowing incremental performance investments.

Photo of AMD's Barcelona quad-core CPU. It consists of about 463 million transistors.
Canneal Characteristics

Working Sets

- Netlist elements
- Netlist

CacheHits

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Huge working sets, communication limited by capacity
Dedup Overview

- Detects and eliminates redundancy in a data stream with a next-generation technique called 'deduplication'
- Enterprise storage kernel (Princeton)
- Input is an uncompressed archive containing various files
- Improved, more computationally intensive deduplication methods
- More cache-efficient serial version
- Pipeline parallelism with multiple thread pools
- Huge working sets, significant communication
Dedup Rationale

- Growth of world data keeps outpacing growth of processing power.
- This data has to be stored and transferred.
- Use cheap resources (processing power) to make more efficient use of scarce resources (storage & bandwidth).
- Already in use in commercial products.

Next-generation storage and networking products already use data deduplication.
Dedup Characteristics

**Working Sets**

- Data chunks
- Hash table

**Cache Hits**

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

**Huge working sets, some communication**
Facesim Overview

- Simulates motions of a human face for visualization purposes
- Computer animation application (Intel + Stanford)
- Input is a face model and a series of muscle activations
- Coarse-grained parallelism, similarities to HPC programs
- Large working sets, some sharing

Source: Eftychios Sifakis et al.

Facesim creates visually realistic animations of a human face
Facesim Rationale

- Video games and other interactive animations require visualization of realistic faces in realtime
- Challenging problem, humans evolved to perceive finest details in a face
- Physical simulation gives excellent results, but is computationally very challenging
- Technology already in use for movie productions (e.g. Pirates of the Caribbean 3)

*Faces are an integral part of contemporary games. Screenshot of Codemasters' “Overlord: Raising Hell” (2008).*
Facesim Characteristics

Working Sets

- Tetrahedra
- Face mesh

Large working sets, some sharing

Cache Hits

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Traffic (Bytes / Instr.)

Cache Size (KB)

Cores

0.00%
1.00%
2.00%
3.00%
4.00%
5.00%
Ferret Overview

- Search engine which finds a set of images similar to a query image by analyzing their contents
- Server application for content-based similarity search of feature-rich data (Princeton)
- Input is an image database and a series of query images
- Pipeline parallelism with multiple thread pools
- Huge working sets, very communication intensive
Ferret Rationale

- Growth of world data requires methods to search and index it
- Noise and minor variations frequently make same content appear slightly different
- Traditional approaches using key words are inflexible and don't scale well
- Computationally expensive

A web interface for image similarity search.
Ferret Characteristics

Working Sets

Cache Hits

Huge working sets, very communication intensive
Fluidanimate Overview

- Simulates the underlying physics of fluid motion for realtime animation purposes with SPH algorithm

- Computer animation application (Intel)

- Input is a list of particles

- Coarse-granular parallelism, static load balancing

- Large working sets, some communication
Fluidanimate Rationale

- Physics simulations allows significantly more realistic animations
- Highly demanded feature for games
- Fluid animation one of most challenging effects
- Already beginning to get used in games

Advanced physics effects are already starting to get used in games: Tom Clancy's Ghost Recon Advanced Warfighter (2006) with (left) and without (right) PhysX effects.
Fluidanimate Characteristics

Working Sets

- Large working sets, some communication

Cache Hits

- Traffic (Bytes/Instr.)
- Traffic (Bytes/Instr.)
- Cache Hits

Cores

- Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Large working sets, some communication
Freqmine Overview

- Identifies frequently occurring patterns in a transaction database

- Data mining application (Intel + Concordia)

- Input is a list of transactions

- Medium-granular parallelism, parallelized with OpenMP

- Huge working sets, some sharing
Freqmine Rationale

- Increasing amounts of data need to be analyzed for patterns
- Applies to many different areas such as marketing, computer security or computational biology
- Requirements for computational processing power virtually unlimited in practice

Frequent Itemset Mining is already used e.g. for e-commerce (Screenshot: Amazon.com).
Freqmining Characteristics

Working Sets

- Transactions
- FP-tree

Cache Hits

Huge working sets, some sharing

Graph showing cache size (KB) vs. miss rate (%), and traffic (Bytes/Instr.) across different cache sizes and cores.
Raytrace Overview

- Uses physical simulation for visualization
- Computer animation application (Intel)
- Input is a complex object composed of many triangles
- Fine-granular parallelism, dynamic load balancing
- Large working sets, little communication, significant data sharing

Source: Stanford University

Native input for raytrace. (10 million polygons)
Raytrace Rationale

- Physics simulations allows accurate visualizations with realistic 3D graphics
- Realistic effects possible without tricks (shadows, reflections, refractions, etc.)
- Simpler development of games at the cost of more expensive computations

Nvidia buys ray-tracing tech company RayScale

May 23, 2008 10:12 AM PDT

Nvidia confirmed Friday that it has acquired RayScale, a small company that develops ray-tracing technology. Financial terms of the deal have not been disclosed.

Ray tracing has been mentioned frequently by Intel over the last six months. An Intel blog titled "Real Time Ray-Tracing: The End of Rasterization?" and later comments by Intel executives that the company is looking at doing ray tracing on its processors set the stage for debate on the viability of ray tracing in mainstream gaming.

PC graphics technology today uses rasterization. (A discussion of ray tracing vs. rasterization.)

Ray Tracing is a technique for rendering three-dimensional graphics with extremely complex light interactions, allowing the creation of transparent surfaces and shadows, for example, with stunning photorealistic results.

Ray tracing is a highly parallel process, and the GPU (graphics processing unit) provides high level of parallelism, according to Nvidia officials speaking at a conference on Thursday. The GPU has special function units that were designed for doing graphics operations that are perfect for ray tracing, said Nvidia Chief Scientist David Kirk.

At the conference, Kirk and RayScale scientists discussed "GPU ray tracing." It's not clear how soon this technology would be used commercially by Nvidia.

RayScale, which provides interactive ray tracing and photo-realistic rendering solutions, says its technologies "dramatically increase the speed and realism at which graphics professionals can produce high quality three-dimensional computer graphics and photorealistic computer images."

RayScale is a product of the decade-long interactive ray-tracing research at the University of Utah, according to RayScale.

At the Intel Developer Forum in Shanghai in April, Senior Intel Vice President Patrick Gelsinger spelled out Intel's

**Major companies have started to invest into ray tracing**
(Source: cnet, May 2008)
Raytrace Characteristics

- Huge working sets containing the whole scene
- Exact working set sizes are data-dependent
- Entire scene is shared among all threads
- Memory bandwidth main issue for good speedups
Streamcluster Overview

- Computes an approximation for the optimal clustering of a stream of data points
- Machine learning application (Princeton)
- Input is a stream of multidimensional points
- Coarse-granular parallelism, static load-balancing
- Medium-sized working sets of user-determined size

Working set size can be determined at the command line
Streamcluster Rationale

- Clustering is a common problem in many fields like network security or pattern recognition.
- Often input data is only available as a data stream, not as a data set (e.g. huge data set that has to be processed under real-time conditions, continuously produced data, etc).
- Approximation algorithms have become a popular choice to handle problems which are intractable otherwise.
Medium-sized working sets of user-determined size
Swaptions Overview

- Prices a portfolio of swaptions with the Heath-Jarrow-Morton framework
- Computational finance application (Intel)
- Input is a portfolio of derivatives
- Coarse-granular parallelism, static load-balancing
- Medium-sized working sets, little communication

Employs Monte Carlo simulation
Swaptions Rationale

- Computerized trading of derivatives has become wide-spread
- High demand for performance: Saving few milliseconds can earn lots of money
- Monte Carlo simulation is a common approach in many different fields
Swaptions Characteristics

Working Sets

- Cache Size (KB)
- Miss Rate (%)

Swaptions

- Medium-sized working sets, little communication

Cache Hits

- Traffic (Bytes/Instr.)
- Cores

Private Reads
- Private Writes
- Shared Reads
- Shared Writes
- True Shared Reads
- True Shared Writes

Medium-sized working sets, little communication
Vips Overview

- Applies a series of transformations to an image
- Media application (Princeton + National Gallery of London)
- Input is an uncompressed image
- Medium-granular parallelism, dynamic load-balancing
- Medium-sized working sets, some sharing

http://www.vips.ecs.soton.ac.uk/
Vips Rationale

- Image processing is one of the most common operations for desktops and workstations.
- Amount of digital photos grows exponentially.
- Professional images can become huge but still need to be handled quickly.
- Benchmark based on real print-on-demand service at National Gallery of London.

The native input set for vips is a picture of the Orion galaxy with 18,000 x 18,000 pixels.
Vips Characteristics

Working Sets

Cache Hits

Medium-sized working sets, some sharing
X264 Overview

- MPEG-4 AVC / H.264 video encoder
- Media application (Princeton + Open Source Community)
- Input is a sequence of uncompressed image
- Coarse-granular pipeline parallelism
- Medium-sized working sets, very communication intensive

http://www.videolan.org/developers/x264.html
X264 Rationale

- Increasing storage and network capacity have made videos popular
- Shift towards digital TV
- MPEG-4 AVC / H.264 is the standard for next-generation video compression

More processing power enables better compression quality

The input frames for x264 were taken from the open source movie “Elephants Dream” (2006).
X264 Characteristics

### Working Sets

- **Macroblocks**
- **Reference frames**

### Cache Hits

- **Traffic (Bytes/Instr.):**
  - **Private Reads**
  - **Private Writes**
  - **Shared Reads**
  - **Shared Writes**
  - **True Shared Reads**
  - **True Shared Writes**

**Medium-sized working sets, very communication intensive**
There aren’t any two workloads with the same combinations
Comparing Program Behavior

- **Question:** How to quantify and compare program behavior

- **A Principle-Component-Analysis (PCA) based Benchmark Analysis Methodology**

- **PCA:** a mathematical procedure (wikipedia)
  
  A set of possibly correlated characteristics
  
  ⇒ A set of uncorrelated principle components (PC)

- **Steps:**
  1. Collect characteristics by simulations or real executions
  2. Run the PCA procedure ⇒ several PCs ⇒ vectors in PCA space
  3. Evaluate the similarity of programs by computing the Euclidean Distance of the vectors in PCA space
  4. Visualize similarity with scatter plots and dendrograms
Redundancy & Similarity

The PARSEC workloads are unique and representative.
PARSEC vs. SPLASH-2

Statistical analysis shows significant differences.

You should expect different results.
Systematic Differences

Benchmark suites cluster in different areas, little overlap

Integrate SPLASH-2 into PARSEC framework

PARSEC and SPLASH-2 complement each other well

Input Set Selection/Evaluation

- **Question:** How to choose input sets with multiple scales to meet various demands, e.g., simulation, real machine?

- **Linear**
  - Linear impact on runtime / loops
  - Typically does not change working set sizes

- **Complex**
  - Frequently affects multiple kernels at the same time
  - Often impacts working set sizes, can change the ratio of the kernel execution time

- **Greedy Heuristic Rules:**
  ① Use linear scaling
  ② Use combination of linear and complex scaling
## Input Set Evaluation

<table>
<thead>
<tr>
<th>Program</th>
<th>Input Set</th>
<th>Problem Size</th>
<th>Problem Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Complex Component</td>
<td>Linear Component</td>
</tr>
<tr>
<td></td>
<td>simsmall</td>
<td>4 cameras, 1,000 particles, 5 layers</td>
<td>4,096 options</td>
</tr>
<tr>
<td></td>
<td>simmedium</td>
<td>4 cameras, 2,000 particles, 5 layers</td>
<td>16,384 options</td>
</tr>
<tr>
<td></td>
<td>simlarge</td>
<td>4 cameras, 4,000 particles, 5 layers</td>
<td>65,536 options</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>4 cameras, 4,000 particles, 5 layers</td>
<td>10,000,000 options</td>
</tr>
<tr>
<td>blackscholes</td>
<td>simsmall</td>
<td>4 cameras, 1,000 particles, 5 layers</td>
<td>1 frame</td>
</tr>
<tr>
<td></td>
<td>simmedium</td>
<td>4 cameras, 2,000 particles, 5 layers</td>
<td>2 frames</td>
</tr>
<tr>
<td></td>
<td>simlarge</td>
<td>4 cameras, 4,000 particles, 5 layers</td>
<td>4 frames</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>4 cameras, 4,000 particles, 5 layers</td>
<td>261 frames</td>
</tr>
<tr>
<td>bodytrack</td>
<td>simsmall</td>
<td>100,000 elements</td>
<td>10,000 swaps per step, 32 steps</td>
</tr>
<tr>
<td></td>
<td>simmedium</td>
<td>200,000 elements</td>
<td>15,000 swaps per step, 64 steps</td>
</tr>
<tr>
<td></td>
<td>simlarge</td>
<td>400,000 elements</td>
<td>15,000 swaps per step, 128 steps</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>2,500,000 elements</td>
<td>15,000 swaps per step, 6,000 steps</td>
</tr>
<tr>
<td>canneal</td>
<td>simsmall</td>
<td>100,000 elements</td>
<td>10 MB data</td>
</tr>
<tr>
<td></td>
<td>simmedium</td>
<td>200,000 elements</td>
<td>31 MB data</td>
</tr>
<tr>
<td></td>
<td>simlarge</td>
<td>400,000 elements</td>
<td>184 MB data</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>2,500,000 elements</td>
<td>672 MB data</td>
</tr>
<tr>
<td>dedup</td>
<td>simsmall</td>
<td>100,000 elements</td>
<td>10,000 swaps per step, 32 steps</td>
</tr>
<tr>
<td></td>
<td>simmedium</td>
<td>200,000 elements</td>
<td>15,000 swaps per step, 64 steps</td>
</tr>
<tr>
<td></td>
<td>simlarge</td>
<td>400,000 elements</td>
<td>15,000 swaps per step, 128 steps</td>
</tr>
<tr>
<td></td>
<td>native</td>
<td>2,500,000 elements</td>
<td>15,000 swaps per step, 6,000 steps</td>
</tr>
</tbody>
</table>

- Four reference input scales
- Both Linear and Complex impacts are included
Most workloads form local cluster $\rightarrow$ linear
Pipelined Programming Model

- Pipelined programming model is the most common model used in products
  - Clean interfaces and modules
  - Parallel programming

<table>
<thead>
<tr>
<th>Workload</th>
<th>Parallelism</th>
<th>Dependency Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pipeline</td>
<td>Data</td>
</tr>
<tr>
<td>bodytrack</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>dedup</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ferret</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>x264</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Significant systematic differences between the two types of programs.
Research by PARSEC

- “Does Cache Sharing on Modern CMP Matter to the Performance of Contemporary Multithreaded Programs?”, Eddy Z. Zhang, Yunlian Jiang, Xipeng Shen, PPoPP, 2010. (Best Paper Award)
- “Characterizing the TLB Behavior of Emerging Parallel Workloads on Chip Multiprocessors”, Abhishekh Bhattacharjee, Margaret Martonosi. PACT 2009, (Best paper Finalist)
- …
Part 2

Working with PARSEC
PARSEC is composed of the framework and packages

- **Framework executable files**
- **Global configuration files**
- **Extended benchmark directory**
- **PARSEC benchmark directory**
- **Each group directory contains one directory per package in that group**
Each package directory is structured as follows:

- **Input archives (optional)**: `[PACKAGENAME]/inputs/
  inst/...`  
- **Build directory for temporary files, one subdirectory per build**: `[PACKAGENAME]/obj/...`  
- **Build installations with one subdirectory per installation**: `[PACKAGENAME]/parsec/`  
- **Local configuration files**: `[PACKAGENAME]/run/`  
- **Source code of package**: `[PACKAGENAME]/src/...`  
- **Run directory for temporary files**: `[PACKAGENAME]/run/...`
Configuration Files

- Global configuration files (in config/ directory of framework):
  - PARSEC main configuration file: parsec.conf
    - 3.0 → package
  - System configurations: [OSNAME].sysconf
  - Global build configurations: [BUILDCONF].bldconf
  - Global run configurations: [INPUT].runconf

- Local configuration files (in parsec/ directory of each package):
  - Local build configurations: [BUILDCONF].bldconf
  - Local run configurations: [INPUT].runconf
Run the following command:

```
parsecmgmt -a status -p parsec
```
Run the following command:

```
parselmgmt -a status -p parsec
```

You should see some information similar to the following one:

```
[PARSEC] Installation status of selected packages:
[PARSEC] parsec.blackscholes:
[PARSEC]   amd64-linux.gcc
[PARSEC]   amd64-linux.gcc-hooks
[PARSEC]   amd64-linux.gcc-tbb
[PARSEC] parsec.bodytrack:
[PARSEC]   amd64-linux.gcc
[PARSEC]   amd64-linux.gcc-serial
[PARSEC] parsec.canneal:
[PARSEC]   amd64-linux.gcc
[PARSEC]   amd64-linux.gcc-pthreads
[PARSEC]   amd64-linux.gcc-serial
```
Run the following command:

```
parssecmgmt -a status -p all
```
parsecmgmt

- A script to help you manage your PARSEC installation
- Can build and run PARSEC workloads for you
- Only there for convenience, you can also do the same tasks manually
- Uses information in configuration files to do its job
- Use the following command to get some help:

  ```bash
cparsecmgmt -h
  ```
Building Workloads

- You can build a PARSEC workload as follows:

  ```
  parsecmgmt -a build -p [suite].[PACKAGE]
  ```

- Flag `-a` specifies the desired action, flag `-p` gives one or more packages
- A package can be a workload, library or anything else that comes with PARSEC and can be compiled
- `parsecmgmt -a info` gives you a list of all available packages
- `Parsecmgmt` will automatically handle dependencies between packages correctly
Q: How do you build workload canneal?
Q: How do you build workload raytrace in parsec suite?
Q: How do you build workload raytrace in splash2x suite?
Q: How do you build package canneal?

A: You can use the following command:

> parsecmgmt -a build -p canneal

[PARSEC] Packages to build: canneal

[PARSEC] [========== Building package canneal ==========
[PARSEC] [---------- Analyzing package canneal ----------
[PARSEC] canneal depends on: hooks
[PARSEC] [---------- Analyzing package hooks -----------
[PARSEC] hooks does not depend on any other packages.
[PARSEC] [---------- Building package hooks -----------
[PARSEC] Copying source code of package hooks.
[PARSEC] Running 'env make':
/usr/bin/gcc -O3 -funroll-loops -fprefetch-loop-arrays
-DPARSEC_VERSION=2.0 -Wall -std=c99 -D_GNU_SOURCE
-D_XOPEN_SOURCE=600 -c hooks.c
ar rcs libhooks.a hooks.o
ranlib libhooks.a
[PARSEC] Running 'env make install':
...
Q: How do you build package canneal?
A: You can use the following command:

```bash
> parsecmgmt -a build -p parsec.canneal
[PARSEC] Packages to build: canneal

[PARSEC] [========== Building package canneal ==========
[PARSEC] [---------- Analyzing package canneal ----------
[PARSEC] canneal depends on: hooks
[PARSEC] [---------- Analyzing package hooks ----------
[PARSEC] hooks does not depend on any other packages.
[PARSEC] [---------- Building package hooks ----------
[PARSEC] Copying source code of package hooks.
[PARSEC] Running 'env make':
/usr/bin/gcc -O3 -funroll-loops -fprefetch-loop-arrays
-DPARSEC_VERSION=2.0 -Wall -std=c99 -D_GNU_SOURCE
-D_XOPEN_SOURCE=600 -c hooks.c
ar rcs libhooks.a hooks.o
ranlib libhooks.a
[PARSEC] Running 'env make install':
...
```
Q: How do you build workload `raytrace` in parsec suite?
Q: How do you build workload `raytrace` in splash2x suite?

A: You can use the following command:

- `parsecmgmt -a build -p parsec.raytrace`
- `parsecmgmt -a build -p splash2x.raytrace`
Suite, Groups & Aliases

- Each package belongs to exactly one group
- `parsecmgmt` also understands aliases
- You can use group names and aliases instead of package names
- Example:
  - `parsecmgmt -a build -p all`
  - `parsecmgmt -a build -p parsec`
  - `parsecmgmt -a build -p splash2x`

- Current Suites are `parsec, splash2x`
- Possible aliases are `kernels, apps, bench, libs, tools and all`
- User-defined aliases `[demo]`
Build Configurations

- Build configurations determine how `parsecmgmt` is to build a package
- Specifies compiler, compiler flags, optimizations, etc.
- Use flag `'-c'` with `parsecmgmt` to select a build configuration
- You should create your own build configurations according to your needs
- Default build configurations are `gcc`, `gcc-hooks`, `gcc-serial` and `icc`
- PARSEC build configurations to enable specific parallelizations are `gcc-openmp`, `gcc-pthreads` and `gcc-tbb`
Q: How do you build workload canneal with build configuration gcc-serial?
Q: How do you build workload canneal with build configuration gcc-serial?

A: You can use the following command:

```bash
> parsecmgmt -a build -p canneal -c gcc-serial
```

[PARSEC] Packages to build: canneal

[PARSEC] [========== Building package canneal ==========
[PARSEC] [---------- Analyzing package canneal ----------
[PARSEC] canneal depends on: hooks
[PARSEC] [---------- Analyzing package hooks ----------
[PARSEC] hooks does not depend on any other packages.
[PARSEC] [---------- Building package hooks ----------
[PARSEC] Copying source code of package hooks.
[PARSEC] Running 'env make':
/usr/bin/gcc -O3 -funroll-loops -fprefetch-loop-arrays
-DPARSEC_VERSION=2.0 -Wall -std=c99 -D_GNU_SOURCE
-D_XOPEN_SOURCE=600 -c hooks.c
ar rcs libhooks.a hooks.o
ranlib libhooks.a
[PARSEC] Running 'env make install':
```
Multiple Builds

- You can have more than one build of every package installed
- Parsecmgmt will create a platform description string to distinguish builds as follows:
  
  \[\text{[ARCHITECTURE]}-\text{[OSNAME]}\cdot\text{[BUILDCONF]}\]

- You can override this string by defining environment variable PARSECPLAT
- PARSEC 2.0 also allows you to append an extension to further distinguish builds
You can see a list of all installed builds if you run:

```
parsecmgmt -a status -p all
```

`Parsecmgmt` will list the platform description strings of all installed builds for each workload:

```
[PARSEC] Installation status of selected packages:
[PARSEC] blackscholes:
[PARSEC]   -no installations-
[PARSEC] bodytrack:
[PARSEC]   -no installations-
...
[PARSEC] canneal:
[PARSEC]   x86_64-linux-gnu.gcc
[PARSEC]   x86_64-linux-gnu.gcc-serial
...
```
Cleanup

- Remove all temporary directories (used e.g. for building):
  
  ```
  parsecmgmt -a fullclean -p all
  ```

- Uninstall a specific installation:

  ```
  parsecmgmt -a uninstall -p [PACKAGE] -c [BUILDCONF]
  ```

- Uninstall everything:

  ```
  parsecmgmt -a fulluninstall -p all
  ```
Running Benchmarks

- You can run a PARSEC benchmark as follows:

  ```
  parsecmgmt -a run -p [PACKAGE] -c [BUILDCONF]
  -i [INPUT] -n [THREADS]
  ```

- Like building workloads, but you can also specify an input and the number of threads

  Flag `\-n` specifies the *minimum* number of threads. The actual number can be higher. You must use other techniques to limit the number of CPUs.

- Default inputs are *test*, *simdev*, *simsmall*, *simmedium*, *simlarge* and *native*
Input Sets

- **Test**
  Execute program, as small as possible, *best-effort* execution path as real inputs

- **Simdev**
  Stresses all machine parts required by larger input sets, same execution path as real inputs

- **Simsmall**
  Like real inputs, runtime ~1s

- **Simmedium**
  Like real inputs, runtime ~5s

- **Simlarge**
  Like real inputs, runtime ~15s

- **Native**
  Like real inputs, runtime ~15min
Q: How do you run the serial version of workload `c anneal` with input `simsmall`?
Q: How do you run the serial version of workload canneal with input simsmall?

A: You can use the following command:

```
> parsecmgmt -r run -p canneal -c gcc-serial -i simsmall
[PARSEC] Benchmarks to run: canneal

[PARSEC] [======== Running benchmark canneal =========]
[PARSEC] Setting up run directory.
[PARSEC] Unpacking benchmark input 'simsmall'.
100000.nets
[PARSEC] Running '...':
[PARSEC] [---------- Beginning of output ----------]
PARSEC Benchmark Suite Version 2.0
Threadcount: 1
10000 moves per thread
Start temperature: 2000
...
[PARSEC] [---------- End of output ----------]
[PARSEC] Done.
```
Log Files

- Parsecmgmt stores all output of builds and runs in log files
- All log files are kept in the log/ directory of the framework
- Naming convention:

  build_[DATE]_[TIMESTAMP].log

  and

  run_[DATE]_[TIMESTAMP].log
Documentation

- Comprehensive documentation shipped with PARSEC
- Full set of man pages available in the `man/` directory
- Add it to the `MANPATH` environment variable to access it (example assumes bash shell):

  ```bash
  MANPATH=${MANPATH}:${PARSECDIR}/man
  ```

- We provide a script `env.sh` which does that for you (see next slide)
- Then you can start browsing the documentation as follows:

  ```bash
  man parsec
  ```
Environment Setup

- You can modify your environment to make the PARSEC tools and its man pages available at the command line (without full path)
- The `env.sh` script in the PARSEC root directory will do that for you
- Source it as follows (example assumes bash shell):
  ```bash
  source env.sh
  ```
- If you use PARSEC a lot you can add that to your login scripts to have it always available
Managing Build Configurations

- Create a new build configuration:
  
  \[ \text{bldconfadd } -n \text{ [NAME]} \]

- In most cases you will want to create a copy of an existing build configuration

- Use flag '\(-c\)' for a hard copy and flag '\(-s\)' for a soft copy

- Delete a build configuration:
  
  \[ \text{bldconfdel } -n \text{ [NAME]} \]

- Use flag '\(-h\)' with both tools to get more detailed usage information
Modifying Build Configurations

- You should adapt build configurations to your needs
- Each build configuration has to define:
  - Default environment variables for makefiles (CC, CXX, CFLAGS, ...)
  - Build tool version numbers (CC_ver, CXX_ver, ...)
  - It should define macro PARSEC_VERSION
- The global configuration files define all parameters, the local ones adapt them and add additional variables as needed by each package
Q: Create a new build configuration `gcc-debug` based on `gcc` that compiles all packages without optimization but with debugging support. Test it on workload `canneal`.
Q: Create a new build configuration `gcc-debug` based on `gcc` that compiles all packages without optimization but with debugging support. Test it on workload `canneal`.

A: First, create a copy of build configuration `gcc`:

```
configadd -n gcc-debug -s gcc
```

Next, edit `gcc-debug.bldconfig` in directory `config/` to use the new flags:

```bash
#!/bin/bash

source ${PARSECDIR}/config/gcc.bldconfig

CFLAGS="\$\{CFLAGS\} -00 -g"
CXXFLAGS="\$\{CXXFLAGS\} -00 -g"
```
Build Information

- Parsecmgmt creates a special file 'build-info' with information about the build in each build installation directory.

- File contains details about build configuration and environment at the time of compilation:
  - Exact location and version of all compilers
  - Compiler flags specified by build configuration
  - Modifications of environment variables

- Makes it a lot easier to figure out what was going on if build configurations were modified.
Q: How did parsecgmnt modify the environment to build the serial version of workload canneal?
Q: How did `parsecmgmt` modify the environment to build the serial version of workload `canneal`?

A: It's in `build-info` for the `gcc-serial` configuration:

PARSEC Compile Information
==========================
Package 'canneal'
Built on Wed May 7 20:24:59 EDT 2008

Configure arguments:  `--prefix=/home/cbienia/parsec/parsec-2.0/pkgs/kernels/canneal/inst/x86_64-linux-linux.gcc-serial`

Environment modifications: version=serial

CC: /usr/bin/gcc
Version: gcc (GCC) 4.1.2 20070626 (Red Hat 4.1.2-14)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
CFLAGS: -O3 -funroll-loops -fprefetch-loop-arrays -DPARSEC_VERSION=2.0 ...
How to add a workload

- Add a “hello” workload
- cd “ext” directory
- create your suite “ext/user”
- Copy template workload to your suite
- change config file
Part 3

Roadmap of PARSEC
Network Workloads

- Network workloads are ubiquitous
- TCP/IP stack is CPU intensive
  - Rule-of-thumb: 1Gbits/sec ~ 1Ghz Pentium CPU
- 10 Gbits are here and CPUs are multicores
  - Need parallelized TCP/IP stack
- No TCP/IP stack in existing benchmarks
Framework

- Goal: A framework easily run network workloads on real machine and simulators

- A user-level, parallelized TCP/IP stack
  - Easy to run on a simulator

- Environment
  - Run client and server workloads together
Approach

- **User-level TCP/IP Stack (u-TCP/IP)**
  - Extract the TCP/IP Stack from FreeBSD kernel
  - Keep u-TCP/IP’s behavior similar

- **Parallelized u-TCP/IP → up-TCP/IP**
  - Use multiple methods to parallelize u-TCP/IP
    - Pipelined model
    - Data parallel
Two Modes

- **Inter-Node**

- **Intra-Node**
GPGPU Workloads

- Many emails asking if we provide GPGPU workloads
- Need Huge Efforts
- Our Plan
  - Encourage people to port PARSEC to GPGPU
  - Submit your GPU-version PARSEC
  - Credits given by the new framework
Part 4

Concluding Remarks
● PARSEC 3.0
  – Release planned for summer 2011

● We need your contribution
  – Network Workloads
  – Porting PARSEC to GPGPU
References


Open Discussion

Where do you think PARSEC should go?

What has to change?

Questions?
The PARSEC Benchmark Suite Tutorial

- PARSEC 3.0 -

by

Yungang Bao, Christian Bienia, Kai Li
Princeton University
PARSEC Hooks

- Write code once, automatically insert into all workloads simply by rebuilding them

- The hooks API functions are called at specific, predefined locations by all workloads

- Implemented as a library

- Comes with several useful features already implemented (see config.h in hooks package)

- Read the man pages for detailed explanations
Enabling PARSEC Hooks

- Define macro ENABLE_PARSEC_HOOKS (and tell the compiler and linker to use the hooks header files and library)

- The following flags work with gcc:
  - For CFLAGS: `-DENABLE_PARSEC_HOOKS`  
    `-I${PARSECDIR}/pkgs/hooks/inst/${PARSECPLAT}/include`
  - For LDFLAGS: `-L${PARSECDIR}/pkgs/libs/hooks/inst/${PARSECPLAT}/lib`
  - For LIBS: `-lhooks`

- The build configuration gcc-hooks does this already by default
PARSEC Hooks API

Application Start

Call to `void __parsec_bench_begin(enum __parsec_benchmark __bench)`

Initialization

Call to `void __parsec_roi_begin()`

Parallel phase

Call to `void __parsec_roi_end()`

Cleanup

Call to `void __parsec_bench_end()`

Application End

<table>
<thead>
<tr>
<th>Parallel Code</th>
<th>Serial Code</th>
</tr>
</thead>
</table>

Region of Interest
PARSEC Hooks Features

- **Measure execution time of ROI**
  Define `ENABLE_TIMING` in `config.h` (enabled by default)

- **Control thread affinity via environment variables**
  Define `ENABLE_SETAFFINITY` in `config.h` (enabled by default, Linux only)

- **Execute Simics “Magic Instruction” before and after ROI**
  Define `ENABLE_SIMICS_MAGIC` in `config.h` (disabled by default, Simics simulations only)
Assisting Simulations with PARSEC Hooks

You can use PARSEC Hooks to eliminate unnecessary simulation time:

Call to `void __parsec_roi_begin()`

- Parallel phase

Call to `void __parsec_roi_end()`

Possible actions:
- Create checkpoint
- Switch from fast-forward to detailed simulation

Possible actions:
- Terminate simulation
- Switch to fast-forward
- Analyze simulation results
Q: Use PARSEC hooks to print out “Entering ROI” if build configuration `gcc-debug` is used. Test it with `canneal`. 
Q: Use PARSEC hooks to print out “I like PARSEC” if build configuration gcc-debug is used. Test it with canneal.

A: Add a print statement to __parsec_roi_begin():

```c
#ifdef ENABLE_MY_OUTPUT
    printf(HOOKS_PREFIX " I like PARSEC\n");
#endif //ENABLE_MY_OUTPUT
```

Define macro for build configuration gcc-debug:

```bash
#!/bin/bash

source ${PARSECDIR}/config/gcc.bldconf

CFLAGS="${CFLAGS} -O0 -g -DENABLE_MY_OUTPUT"
CXXFLAGS="" ${CXXFLAGS} -O0 -g -DENABLE_MY_OUTPUT"
```
Q: Use PARSEC hooks to print out “I like PARSEC” if build configuration gcc-debug is used. Test it with canneal.

A: Remove any existing installations of gcc-debug:

```
parscmgmt -a uninstall -c gcc-debug -p hooks canneal
```

Build and run canneal:

```
parscmgmt -a build -c gcc-debug -p canneal
```
```
parscmgmt -a run -c gcc-debug -p canneal
```