Benchmarking for High Performance Systems and Applications

Erich Strohmaier

NERSC/LBNL

Estrohmaier@lbl.gov

HPC Reference Benchmarks

- To evaluate performance we need a frame of reference in the performance space
- This can be established by a set of benchmarks
- All we have right now is peak performance and Linpack
- We need to better understand what are the critical aspects of algorithms which determine their performance

General Approach

- Develop a new characterization of codes focusing on performance aspects
- Avoid using any specific hardware models or concepts for this characterization
- Develop synthetic performance probes and benchmarks testing these characteristics
- Relate benchmark performance with code performance
- Focus on data-access (initially)

Benchmark Requirements

Benchmarks should be designed to be useful:

- For a significant period of time
- Architecture independent
- Across several generations of different HPC architectures
- Useful for different application domains
- System and generation scalable code and problem definition
- Have stable and well understood performance characteristics

Scalable Definition

- A benchmark useful for the ‘TOP500 class’ of systems for 10 years must bridge a range of **100,000** in system size!
- Benchmarks should utilize resources on a variety of system sizes fully
- Runtimes stable for several generations of systems
  - Eliminate the implications of algorithmic complexity of real applications

Characterizing Performance

- Characterize performance behavior of applications and algorithms independent from hardware!
  - "Time to solution = f(Algorithmic Complexity, Data Access Characteristics, Structure of Operations)"

Data Access Characteristics

What do we want to capture?

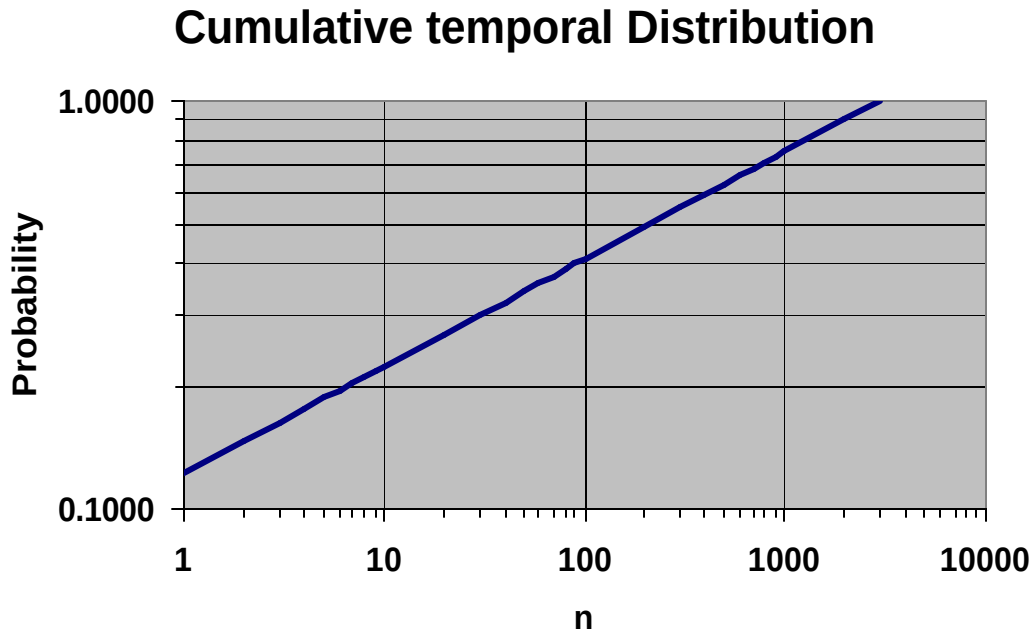
- Re-use of data by modern algorithm for improving locality – *Temporal locality*
  - hierarchical block-structured or recursive algorithms
- Indexed memory access - *Regularity*
  - sparse, irregular, and/or adaptive codes
  - not stride 1 (n)
- Limitations of loop-length - *Granularity*
  - Due to data-dependencies, communication, etc

Temporal Locality

- How can we *quantitatively* describe data re-use?
- Look at temporal distribution function
  - The probability distribution of how long ago I last used a data item
    - At every access I have a *f(t)%* probability to hit a location I have visited within the last *t cycles*.
  - (similar to stack distribution, stack distance)
- Approximate it by a simple function
  - For recursive algorithms the cumulative temporal distribution function should be self-similar and scale-invariant
  - Power Function Distribution

Power Distribution

- Characterized by one number
  - *Slope* related to the 'Re-use' factor
- Concept does not use hardware concepts such as '*cache*'
- Distribution function is problem size and scale invariant



Regularity

- A mapping of the data structure to the address space which permits stride 1 (n) access exposes regularity.
- Re-mapping during execution might be necessary for many algorithms to expose regularity.
  - This form of ‘dynamic’ regularity has associated re-mapping costs (gather-scatter operations).
  - This type of (“irregular”) data access becomes more and more important and is usually not avoidable.
  - If present in a code it is likely to become the performance bottleneck.
  - “*Our focus*”

Granularity

Limitation of loop-length due to data-dependencies.

- The amount of pre-computable addresses (vector-length)
  - Access can be irregular (‘indirect’) or
  - Regular (‘strided’)
- We focus on indirect as it becomes more important and represent more of a lower-bound for achievable performance.
- Granularity becomes very important for parallel version with explicit communication
  - It (severely) limits message sizes

Synthetic Benchmark Probe

- Steady-state throughput measurements.
  - Warm caches etc.
- Non-uniform random memory access .
  - Power-function as temporal distribution function .
- Granularity
  - Vector length for pre-computed addresses and
  - Organization of communication.
- Use indexed (“irregular”) data access to measure a lower bound for performance.
- We have only 2 parameters so far (Good!).

Status and Future

- Implementing several test-codes.
  - Which kernel – DAXPY?
  - How many different index vectors?
    - Impacts also data structures and regularity.
- Extending the concept to parallel systems.
  - Details of the random process – homogeneous or inhomogeneous memory-access?
  - Data-mapping – organized or pseudo-random?
- Determine the re-use factors and granularities for actual codes (paper and pencil).
- Need to test the correlation between benchmark probe performance and code performance for the same re-use factors and granularities.

The Performance Evaluation Research Center (PERC)

David H Bailey

NERSC, Lawrence Berkeley National Laboratory

dhbailey@lbl.gov

PERC Overview

- An “Integrated Software Infrastructure Center” (ISIC) sponsored under DoE’s SciDAC program.
- Approximately $2.4 million per year for 4 years.
- Four DoE laboratories.
- Four universities.
- Mission:
  - Develop a science of performance, and engineer tools for performance analysis and optimization.
  - Focus on large, grand-challenge calculations, such as in SciDAC application projects.

PERC Participants

Laboratories:

- LBNL (Bailey, Strohmaier)
- LLNL (Quinlan, de Supinski, Vetter)
- ORNL (Worley, Dunigan)
- ANL (Hovland, Norris)

Universities:

- Univ of Tennessee (Dongarra)
- Univ of Illinois (Reed)
- Univ of Maryland (Hollingsworth)
- UCSD Supercomputer Center (Snavely)

Thesis

For the foreseeable future, time to solution will be dominated by a code's ability to effectively utilize the memory hierarchy, including local and distributed memory.

Questions

- How can we best measure the memory hierarchy behavior of a particular code on a particular system?
  - Better benchmarks.
- Can we construct accurate models of performance, based on data that is easily obtained?
  - Performance monitoring tools.
- Can we accurately project the performance of a future version of a code on a future system?
  - Performance modeling and analysis.
- If we determine that a given code is running sub-optimally, can we facilitate the necessary changes to improve performance?
  - Software tools to automatically or semi-automatically optimize user codes.

Benchmarks

Discipline-specific benchmarks:

- Polished, concise versions of real user codes.
- Represent strategic application areas.
- Serve as test-bed for tools and methods.

Kernel benchmarks:

- Extracted from real codes.
- Reduce complexity of analyzing full-size benchmarks.

Low-level benchmarks:

- Measure key rates of data access at various levels of memory hierarchy.
- Measure issue rates of functional units, network bandwidth and latencies, costs of TLB misses, OS context switches, I/O rates, etc.
- e.g MAPS (Allan Snavely, SDSC)

Performance Tools

- **SvPablo:** A graphical environment for instrumenting application source code and browsing performance data.
- **Sigma++:** Uses runtime information to extract a detailed representation of an application's memory reference pattern.
- **PAPI:** A unified cross-platform tool to collect hardware performance monitor data.
- **Dyninst:** Provides a machine independent interface to permit the creation of tools and applications that use runtime code patching.
- **Repository in a box:** A toolkit to facilitate the construction and management of performance data collections.

Work is being done by Dan Reed (U Illinois), Jeff Hollingsworth (U Maryland), Jack Dongarra (U Tennessee) and others.

Modeling Techniques

- Analytic Phase Modeling (Worley)
  - Performance models for phases of the execution based on straightforward counts of operations from source code.
- Application Signatures (De Supinski, Vetter, Snavely)
  - Characterize fundamental aspects of an application, independent of the machine where it executes.
- Machine Signature (Snavely)
  - Characterize fundamental aspects of a machine, independent of the applications executing on it.
  - Performance-predictive convolutions: Combines application and machine signatures.
- And others

Modeling Techniques

- Performance algebra (Snavely and Reed):
  - Constructs parameters for various analytic models.
- Black-box performance modeling (Strohmaier):
  - Combines generic algebraic functions with results from basic performance measurements.
- Back-fitting and statistical methods (Strohmaier, Vetter):
  - Uses sophisticated statistical methods based on empirical data.
- Performance bound modeling (Hovland).

Each of these schemes has potential advantages and disadvantages – we need to try and compare them on numerous specific problems.

Performance Optimizers

ROSE (Quinlan, Hovland):

- Extensible mechanism for compile-time optimization.

New Harmony (Hollingsworth):

- Automatically adapts performance based on runtime observations of machine, operating environment and dataset.

ATLAS and AEOS (Dongarra):

- Self-tuning linear algebra software.
- Extension of ATLAS concept to more general applications.

Performance portability programming (Worley):

- Techniques for near-optimal performance across systems.

Performance assertions (Vetter):

- User-specified run-time tests that possibly change the course of the computation depending on results.

Summary

- Achieving optimal performance on HPC systems has compelling economic and scientific rationales.
- Performance is poorly understood – in depth-studies do not exist except in a handful of cases.
- PERC will pursue “performance science” and “performance engineering”, including improved benchmarks, monitoring tools, modeling techniques, and optimizers.

Evaluation of Leading Parallel Architectures for Scientific Computing

Leonid Oliker

Future Technologies Group

NERSC/LBNL

www.nersc.gov/~oliker

Overview

- Most real-life applications are complex, irregular, and dynamic.
- Generally believed that unstructured methods will constitute significant fraction of future high-end computing.
- Evaluate existing and emerging architectural platforms in the context of irregular and dynamic applications.
- Examine the complex interactions between high-level algorithms, programming paradigms, and architectural platforms.

Ongoing Research: Architectural Alternatives

- Observation: Current cache-based supercomputers perform at a small fraction of peak for memory intensive problems (particularly irregular ones).
- Performance directly related to how well memory system performs.
- But “gap” between processor performance and DRAM access times continues to grow (60%/yr vs. 7%/yr).
- Tighter integration of processor and memory is necessary:
  - VIRAM (PIM with on-chip DRAM using vector technology)
  - Imagine (Stream Technology)
- Evaluate use of VIRAM and IMAGINE chips as a building block for high performance machines.
- Examine memory-intensive benchmarks.

New Evaluation Project: Modern Parallel Vector Sys.

- Vector Architectures: SX6, SV2, and ESS
- We plan to study key factors of modern parallel vector systems, including runtime, scalability, programmability, portability, and memory overhead while identifying potential bottlenecks
- Application Domains we plan to examine :
  Climate (CCM3), Fusion (GTC), Material Science (Paratec),
  Molecular Dynamics (NAMD), Astrophysics (Madcap),
  Fluid Dynamics (Overflow)
- What fraction of scientific codes suitable for these architectures?
  What is the best programming paradigm?
  What are required algorithmic modifications?
  What are scalability limiting factors?
  What are migration issues in terms of performance portability?