140 attendees

- Academia (staff & PhD): 50%
- Industry: 14%
- Other: 4%
- Students: 32%
Big Data is not about the data but about the analytics.
Big Data is not about the data but about the analytics. They are algorithms that consume or produce large sets of heterogeneous data, possibly at high frequency. EU calls it $V^3 = \text{Volume, Variety, Velocity}$.
High-Performance Computing is no longer about computing, it is about moving data fast

this is the **Von Neumann** bottleneck, memory wall and dark silicon are its effects
High-Performance Computing is no longer about computing, it is about moving data fast. This is the Von Neumann bottleneck, memory wall, and dark silicon are its effects.

<table>
<thead>
<tr>
<th>Component</th>
<th>Energy per Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>4.8 nJ/word</td>
</tr>
<tr>
<td>Optimized DRAM core</td>
<td>128 pJ/word</td>
</tr>
<tr>
<td>MIPS 64 core</td>
<td>400 pJ/cycle</td>
</tr>
<tr>
<td>11 nm 0.4 V core</td>
<td>200 pJ/op</td>
</tr>
<tr>
<td>45 nm 0.8 V FPU</td>
<td>38 pJ/Op</td>
</tr>
<tr>
<td>SERDES I/O</td>
<td>1.9 nJ/Word</td>
</tr>
<tr>
<td>20 mV I/O</td>
<td>128 pJ/Word</td>
</tr>
<tr>
<td>LPDDR2</td>
<td>512 pJ/Word</td>
</tr>
<tr>
<td>1 cm / high-loss interposer</td>
<td>300 pJ/Word</td>
</tr>
<tr>
<td>0.4 V / low-loss interposer</td>
<td>45 pJ/Word</td>
</tr>
<tr>
<td>On-chip/mm</td>
<td>7 pJ/Word</td>
</tr>
<tr>
<td>TSV I/O (ESD)</td>
<td>7 pJ/Word</td>
</tr>
<tr>
<td>TSV I/O (secondary ESD)</td>
<td>2 pJ/Word</td>
</tr>
</tbody>
</table>

Various Sources
High-Performance Computing is no longer about computing, it is about moving data fast.

This is the Von Neumann bottleneck, memory wall and dark silicon are its effects.

This is about parallel programming models.

Programming Model: neither a language nor a library.
SPEEDUP WITH N PROCESSORS

\[ S = \frac{(N T_F + T_\oplus)}{\log_2(N) + T_F} \]
MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program’s execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable; a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google’s clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc. to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical “record” in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis...
TOWARDS PARALLEL PROGRAMMING
BY TRANSFORMATION: THE FAN SKELETON FRAMEWORK*

M. ALDINUCCI\textsuperscript{a}, S. GORLATCH\textsuperscript{b,1}, C. LENGAUER\textsuperscript{b}
and S. PELAGATTI\textsuperscript{a}

\textsuperscript{a}Dipartimento di Informatica, Università di Pisa, 40 Corso Italia,
I-56125 Pisa, Italy; \textsuperscript{b}Fakultät für Mathematik und Informatik,
Universität Passau, D-94030 Passau, Germany

\textbf{FAN} programs obey the \textit{single-assignment} principle, meaning that
there can be only one definition for each variable. All output variables
are defined in the program's body. Each equation is a \textit{let} expression.

Expressions \((e)\) can be constructed from constants, function applications or
skeleton functions such as \textit{map}, \textit{reduce}, etc.,
\begin{align*}
e & = c | x | e | e | \lambda x . e | E_1 \hspace{1em} E_2 | E_3 \hspace{1em} e \hspace{1em} e \\
E_1 & = \text{map} | \text{map} | \text{reduce} | \text{scanL} | \text{copy} | \text{split} | \text{part} | \text{rearrange} \\
E_2 & = \text{pair} | \text{proj} \\
E_3 & = \text{loopfor} | \text{loopwhile} | \text{looperp}
\end{align*}

The scope of each variable definition extends across all subsequent
definitions in the same body. At the end of the body, we can specify local
definitions using a \textit{where} clause. Names of \textbf{FAN} programs can be used as
functions in expressions.

\begin{verbatim}
inner.product (in a, b : Array n Scalar, out c : Scalar) {
  t = map (*) (pair (a, b));
  c = reduce (+) t;
}
\end{verbatim}

\textbf{FIGURE 1} A \textbf{FAN} program to compute the inner product of two vectors.
Systematic Efficient Parallelization of Scan and Other List Homomorphisms

Sergei Gorlatch*
University of Passau, D-94030 Passau, Germany

Abstract. Homomorphisms are functions which can be parallelized by the divide-and-conquer paradigm. A class of distributable homomorphisms (DH) is introduced and an efficient parallel implementation schema for all functions of the class is derived by transformations in the Bird-Meertens formalism. The schema can be directly mapped on the hypercube with an unlimited or an arbitrary fixed number of processors, providing provable correctness and predictable performance. The popular scan function (parallel prefix) illustrates the presentation: the systematically derived implementation for scan coincides with the practically used "folklore" algorithm for distributed memory machines.

1 Introduction

This paper deals with formal design of parallel programs. We advocate that the issues of correctness and performance should and can be addressed during the design/derivation process, rather than as an afterthought.

A non-distributable example is the Bird-Meertens Formulas (BMF) [5].

3 Distributable Homomorphisms

We introduce a specific class of DH in the form of the combine operator (\(\circ\))

Definition 3. For two binary lists \(u\) and \(v\), the combine operator \(\circ\) can list \(u \circ v = zip(\circ)(u,v)\)

We write \(\circ \downarrow \circ\) for the following:

\[\circ \downarrow \circ[a] = [a]\]

\[\circ \downarrow \circ(x \oplus y) = ((\circ \downarrow \circ) x \oplus (\circ \downarrow \circ) y)\]

Definition 4. Function \(h : \#\) is \(\circ \downarrow \circ\) for some \(\oplus\) and \(\circ\).

Fig. 1. A general homomorphism (left) and a distributable homomorphism (right), computed on a concatenation of two lists

The "distributed reduction": \(red(\circ)x = [red(\circ)x, red(\circ)x, \ldots, red(\circ)x]\)

is obviously a homomorphism with the combine operator:

\[R_1 \circ R_2 = zip(\circ)(R_1, R_2) \oplus zip(\circ)(R_1, R_2)\]
1.4 Map

The operator \( \cdot \) (pronounced ‘map’) takes a function on its left and a list on its right. Informally, we have

\[
* \, [a_1, a_2, \ldots, a_n] = [f(a_1), f(a_2), \ldots, f(a_n)]
\]

Formally, we specify \( * \) by three equations

\[
\begin{align*}
* \, [] & = [] \\
* \, [a] & = [f(a)] \\
* \, (x \cdot y) & = (f \cdot x) \cdot (f \cdot y)
\end{align*}
\]

Thus, for \( f : \alpha \rightarrow \beta \) the function \( * \) is a homomorphism from \( ([\alpha], \cdot, []) \) to \( ([\beta], \cdot, []) \). This function is a homomorphism on bags and sets as well as lists.

An important property of \( \cdot \) is that it distributes (backwards) through functional composition:

\[
(f \cdot g) \cdot x = (f \cdot (g \cdot x))
\]

This fact will be referred to as the (\( \cdot \) distributivity) rule. Its use in calculations is so frequent that we shall sometimes employ it without explicit mention.

1.5 Reduce

The operator \( / \) (pronounced ‘reduce’) takes a binary operator on its left and a list on its right. Informally, we have

\[
/ \, [a_1, a_2, \ldots, a_n] = a_1 \cdot a_2 \cdot \ldots \cdot a_n
\]

Formally, we specify \( / \), where \( \cdot \) is associative, by three equations

\[
\begin{align*}
/ \, [] & = \text{id}_\cdot \\
/ \, [a] & = a \\
/ \, (x \cdot y) & = (/ \, x) \cdot (/ \, y)
\end{align*}
\]

If \( \cdot \) is commutative as well as associative, then \( / \) can be applied to bags; and if \( \cdot \) is also idempotent, then \( / \) can be applied to sets.

If \( \cdot \) does not have an identity element, then we can regard \( / \) as a function of type \( / : [\alpha]^+ \rightarrow \alpha \). Alternatively, we can invent an extra value and
Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose
INVITED TALKS (9.00-11.15)

- **M. Drocco & C. Misale.** One Programming Model to dominate them all — Hadoop, Spark, Flink, Storm, Tensorflow: Instructions for Use

- **Y. Park.** OpenPOWER, Big Data and HPC

- **M. Torquati.** FastFlow, a programming model for FastData

- **K. S. Candan.** Assured and Scalable Data Engineering — Challenges and Opportunities

- **F. Bonchi.** Big Data: what’s really new, what is not, and the need for new algorithms

- **C. Nardone.** Fast Data = Big Data + GPU acceleration
IBM SUPPORT AND AWARDS

• M. Drocco and C. Misale UNITO PhD students, intern at IBM T.J. Watson for 6 months (2015). M. Drocco has been also intern at IBM Research Dublin (2013)

• M. Drocco is a recipient of a 2015 IBM Scholarship award (with a $20000 price)

• M. Aldinucci is a recipient of a 2015 IBM Faculty award (with a $20000 price)
PANEL (9.00-11.15)

Two questions, many opinions …

IBM, NVidia, Ernst&Young, INFN, Oracle Labs, Italtel, List group, Torino wireless, Concept Reply, Noesis Solutions, Nuance, RAI – CRIT, and …

you
COFFEE BREAK AND LUNCH

• Coffee break: ~ 11.15

• Free access to everybody

• Lunch ~13.00 - 1st floor

• Free access to everybody explicitly registered for lunch or participating to afternoon session
JAM SESSION (14.00-16.00)

Jam session is a musical event where musicians play (i.e. "jam") by improvising without extensive preparation or predefined arrangements.

Jam sessions are often used by musicians to develop new material, find suitable arrangements, or simply as a social gathering.
PANEL

FastData@UNITO
C3S@UNITO: COMPETENCY CENTRE ON SCIENTIFIC COMPUTING

A level II center of UNITO with a mission …

… research
Coordinator, grant holder

Technical design proposal, 1M€ EU tender proposal

Design team (5 people)

End-user requirements, expectations, sustainability plans, technical constraints, ...

Life Sciences
Biotechnology
Nanostructure
Mathematics
Chemistry

Computer Science

Physics & INFN

Hosting, infrastructural management

Humanistic
Statistics
Multimedia

Oncology
Geology
Biology
Political Sciences

Business

Molecular Systems Bio
Veterinary

Nanostructure
OCCAM@UNITO

~1K cores HT, ~16K CUDA cores, ~1PB archive, 
~320 TB high-performance scratch storage

1PB (180x6TB) LUSTRE archive
320TB (80x4TB) LUSTRE scratch

32 nodes light
2x12 HT 128 GB RAM

4 nodes fat
4x12 HT 768 GB RAM

4 nodes GPU
4x12 HT 768 GB RAM
+ 2 x K40 GPUs


• **REPARA** (EC-STREP, 7th FP): Reengineering and Enabling Performance And poweR of Applications (2013, 36 months, total cost 3.5M €).


• **cHiPS**et (EC-COST Action IC1406): High-Performance Modelling and Simulation for Big Data Applications (2015, 48 months, total cost 500K €).

• **HiPEAC3** (EC-NoE, 7th FP) European Network of Excellence on High Performance and Embedded and Compilation (2016, 48 months)

• **IBM** Joint Study Agreement: Spark optimisation (est. 2015)

• **Noesis** Solutions: Parallel machine learning techniques for engineering (est. 2015)

• **A3CUBE**: FastFlow/PGAS with in memory fabric (est. 2014)

• **NVidia Corp**: CUDA Research Center at University of Torino (est. 2013)
• Marco Aldinucci – UNITO (moderator)
• Fabrizio Antonelli – Ernst&Young
• Stefano Bagnasco – INFN
• Daniele Bonetta – Oracle Labs
• Raffaele Calogero – MBC
• Paolo Secondo Crosta – Italtel
• Cristian Dittamo – List group
• Chiara Ferroni – Torino wireless
• Cristina Chesta – Concept Reply
• Marco Panzeri – Noesis Solutions
• Yoonho Park – IBM
• Daniele Sereno – Nuance
• Luca Vignaroli – RAI – CRIT

• **Framing** What is the relation between your company and Big Data

• **Volume&Velocity** Is throughput and latency important for your applications or frameworks? Which is your data+computation models (batch, stream, …)? Does scalability and speedup likely to give you a competitive advantage?

• **Interactivity&Viz** How Big Data is going to be visualised or navigated? Does it require additional computation? Do we already know a suitable class of abstraction and algorithms?