Faster FAST
Multicore Acceleration of Streaming Financial Data

Virat Agarwal, David A. Bader, Lin Dan, Lurng-Kuo Liu,
Davide Pasetto, Michael Perrone, Fabrizio Petrini

Financial Market

“Finance can be defined as the art and science of managing money”

L.J. Gittman. Principles of Managerial Finance

More precisely, finance studies the allocation and use of money and other assets and the management of these using:

- Risk management
- Portfolio Optimization
- Financial Engineering
- Algorithmic Trading

Overall Requirements

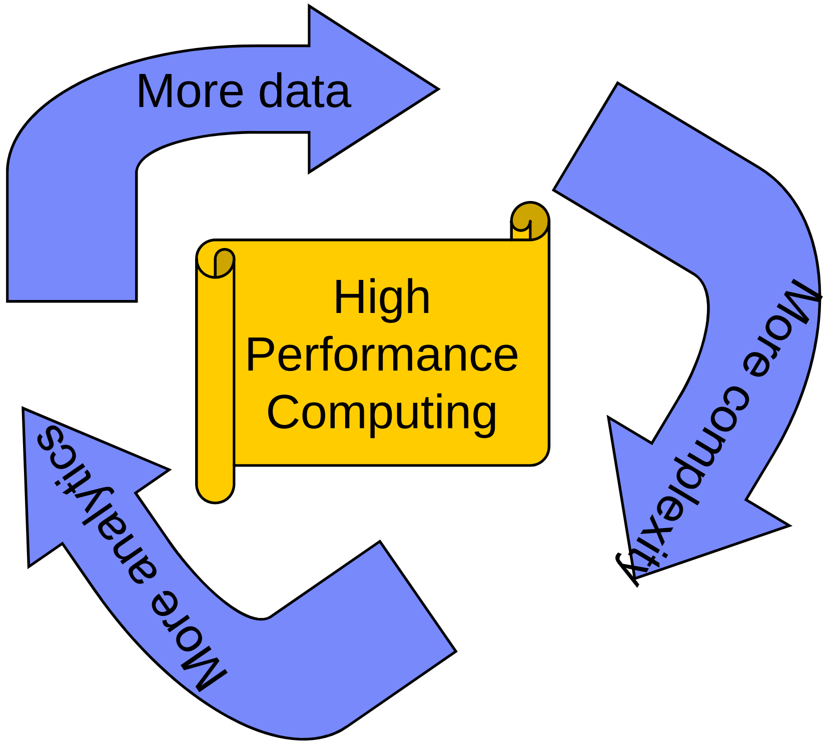
Financial institution perform operations on financial markets by exchanging data with stock exchanges and other neutral contact points

Financial institutions that obtain higher wins are those that can manage:

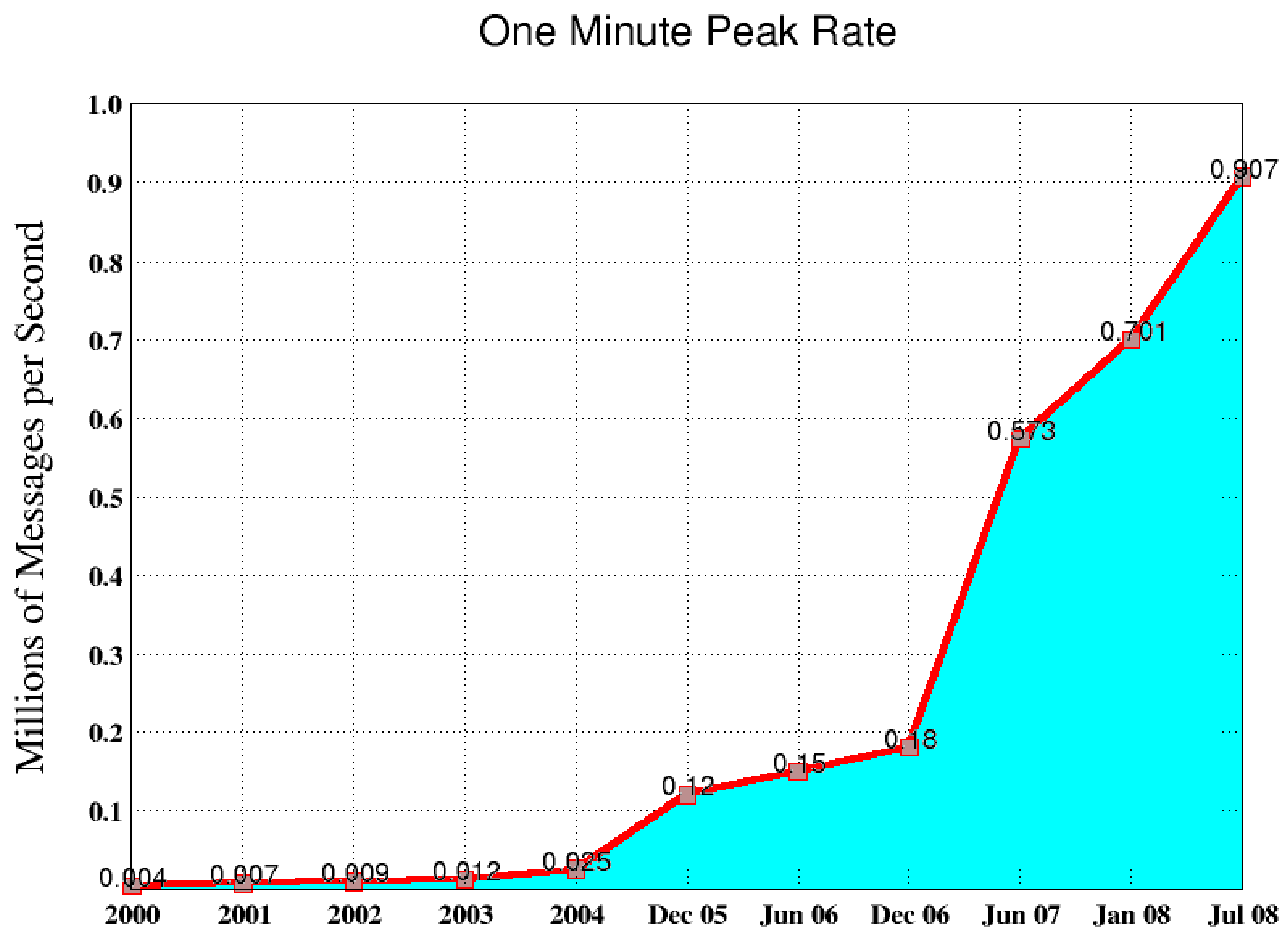
I. higher data volumes, reducing latency;
II. greater data complexity, analyzing more information; and
III. more challenging data analysis, for better data insight

Data Handling

These requirements in data and analytics constitute real challenges and drive the need for greater compute power within financial institutions.



Skyrocketing Data Rates



The Challenge

- End to End low latency is critical, especially with the increasing competition and diminishing profit margin.
- **Few milliseconds gap** in latency is enough to make the difference and will determine who wins and who loses
- The first one to identify a trading opportunity generally does not leave much behind for whoever is next in the line

HPC for Wall Street

Many of the technologies commonly used in high performance computing are appearing in financial institutions data centers:

- Advanced communication fabric such as Infiniband or 10 Gigabit Ethernet,
- low-latency OS-bypass communication protocols,
- hardware accelerators (GPGPUs, FPGAs, etc.),
- latest generation of multi-core processors

All these technologies are becoming essential building blocks to meet the increasing demands of the financial markets.

HPC for Wall Street

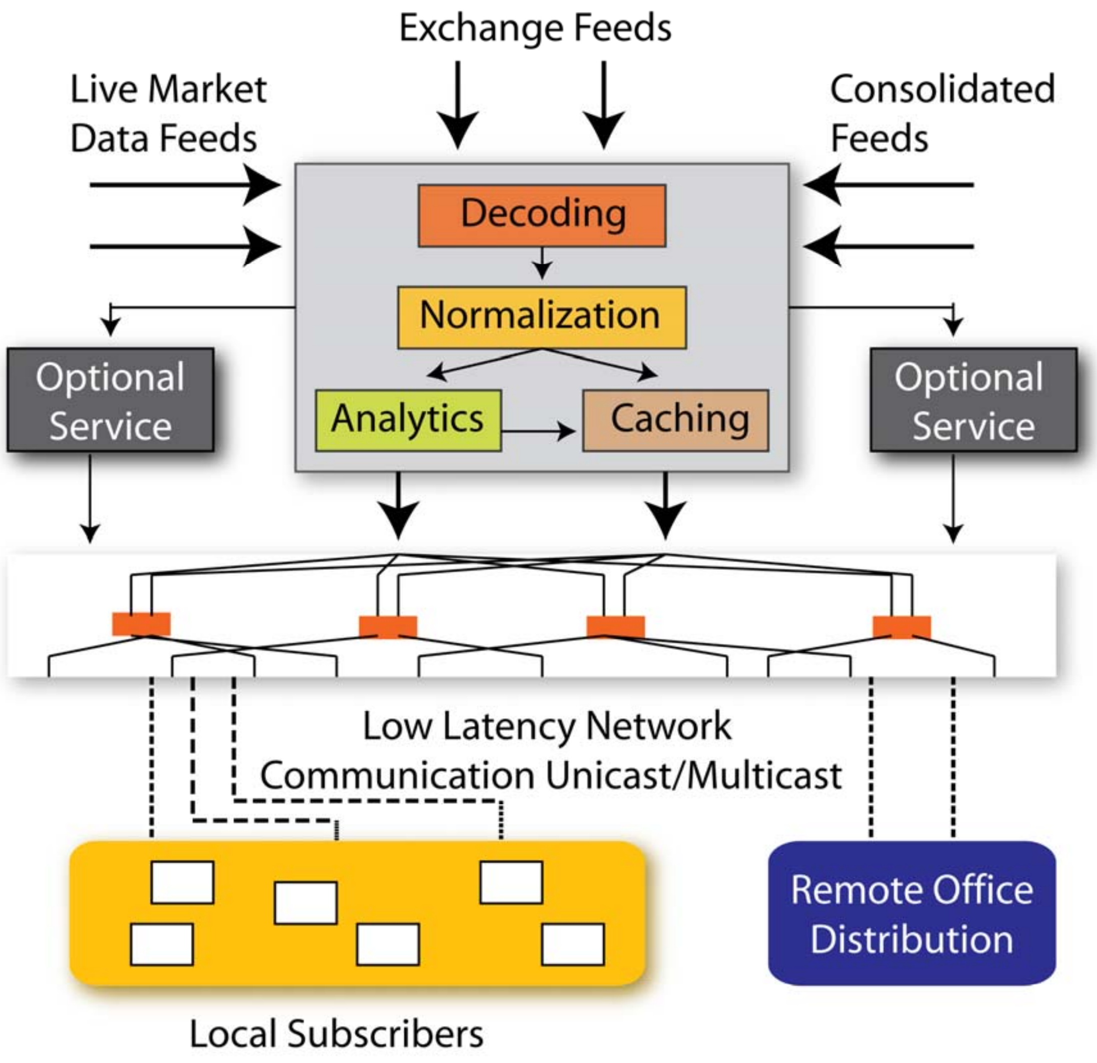
Financial Institutions are also encountering problems typical of high performance computing applications:

- the complexity of developing,
- testing and
- validating parallel software.

Other essential requirement are:

- to reduce power consumption in the processing units (often “co-located” near the data feeds to minimize the communication latency), and
- to reduce floor space requirement, which typically comes at a high premium.

The Ticker Plant



Our contribution

Our objective is to “*Design a ultra-low latency and high-throughput engine for* ***decoding real-time market data feeds***” by:

- Studying the data ingestion and normalization “problem”,
- Utilizing modern high performance *off-the-shelf* CPU,
- Applying both high performance computing techniques and algorithmic innovation,
- Analyzing the performance characteristics of the used CPU architectures

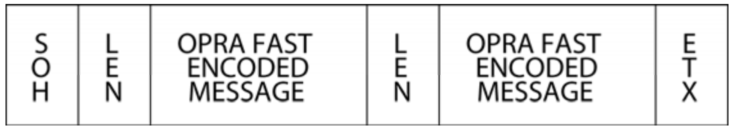
In this study we focus on OPRA (Options Pricing Reporting Authority) feeds.

OPRA : Introduction

- Options Pricing Reporting Authority (OPRA) disseminates information on transactions occurred in Options market.
- Data rates peaking over a million messages/sec, growing exponentially.
- Transactions are disseminated using a compressed format to reduce bandwidth requirements.
- Existing trading solutions usually employ specialized hardware (FPGA) and/or several CPU cores perform decoding and normalization.

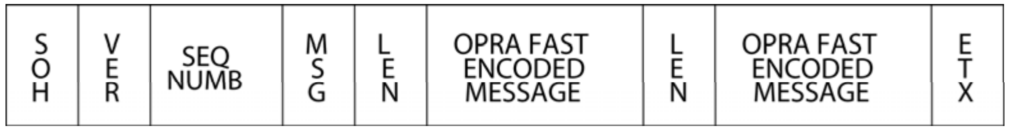
OPRA Message Format

Version 1.03



- Each OPRA packet consists of :
  - SOH and ETX to denote start and end of packet
  - Multiple OPRA messages containing
    - Message Length (1 byte)
    - Presence Map (PMAP) field (<= 8 bytes)
    - Data fields (<=60)
- The order of fields is fixed.

Version 2.0



- The protocol specification changes rapidly, requiring the need of a portable yet scalable high-performance solution.

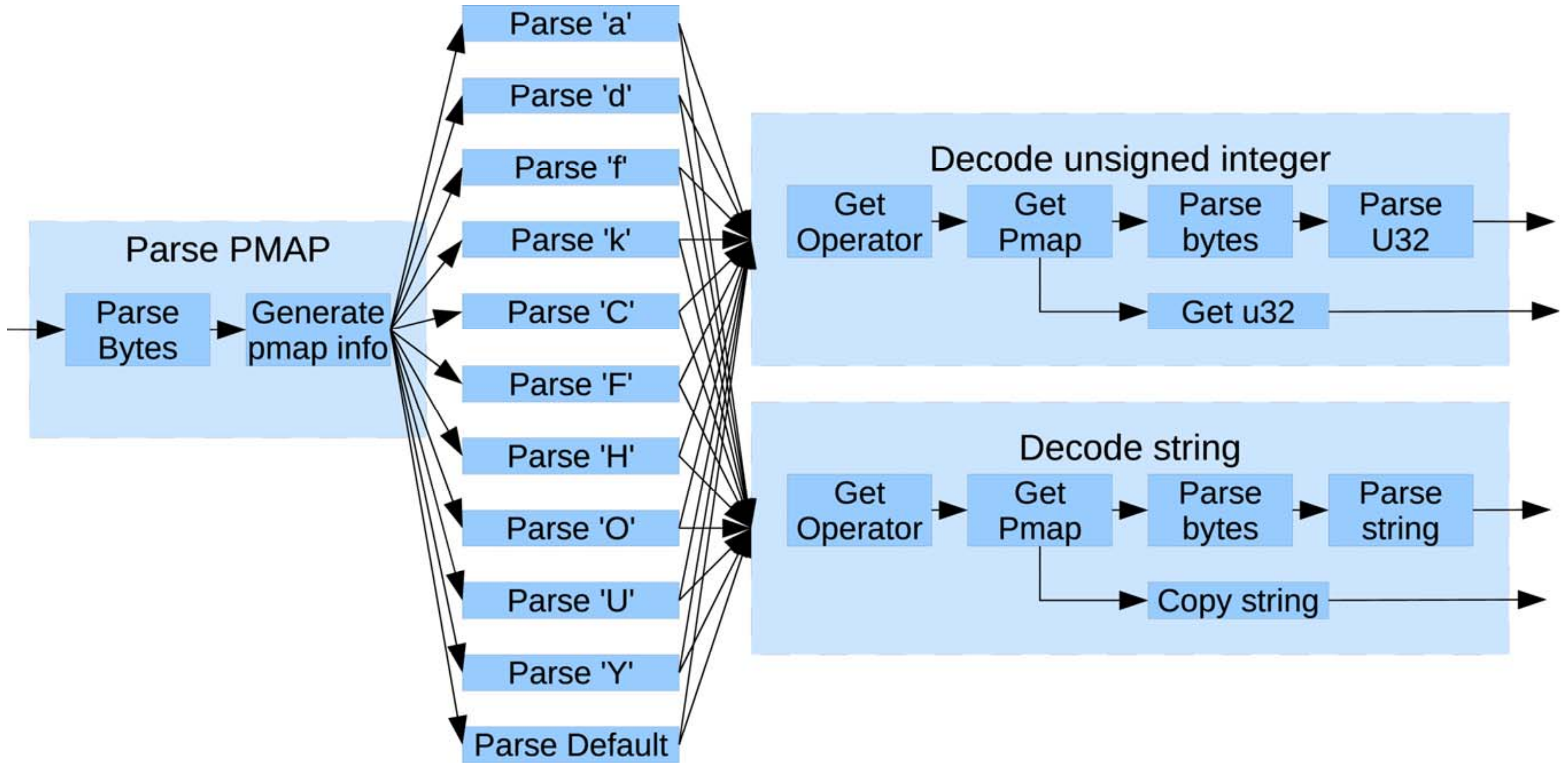
OPRA Message Encoding

- **Stop bit :** The most signification bit of each byte denotes whether this byte marks the end of the field.
  - e.g. : … [0x03 0x50 0x8A] [0x55 0x7F 0x4A 0xA0] …

- **PMAP encoding :** Bit mask that gives information on what fields are present in the message.
  - Data field corresponding to each bit is present in the OPRA specification manual.
  - e.g. : [0x01 0x50 0x80] : specifies that fields 8, 10, 12 are present in the message (1 in last bit is stop bit).

- **Field encoding :**
  - Actions corresponding to each field is provided in the protocol specification.
    - Field is either unsigned integer/string
    - Each field is either copied/incremented/invalid.
    - Further field conditions based on category and type of message (given by data fields).

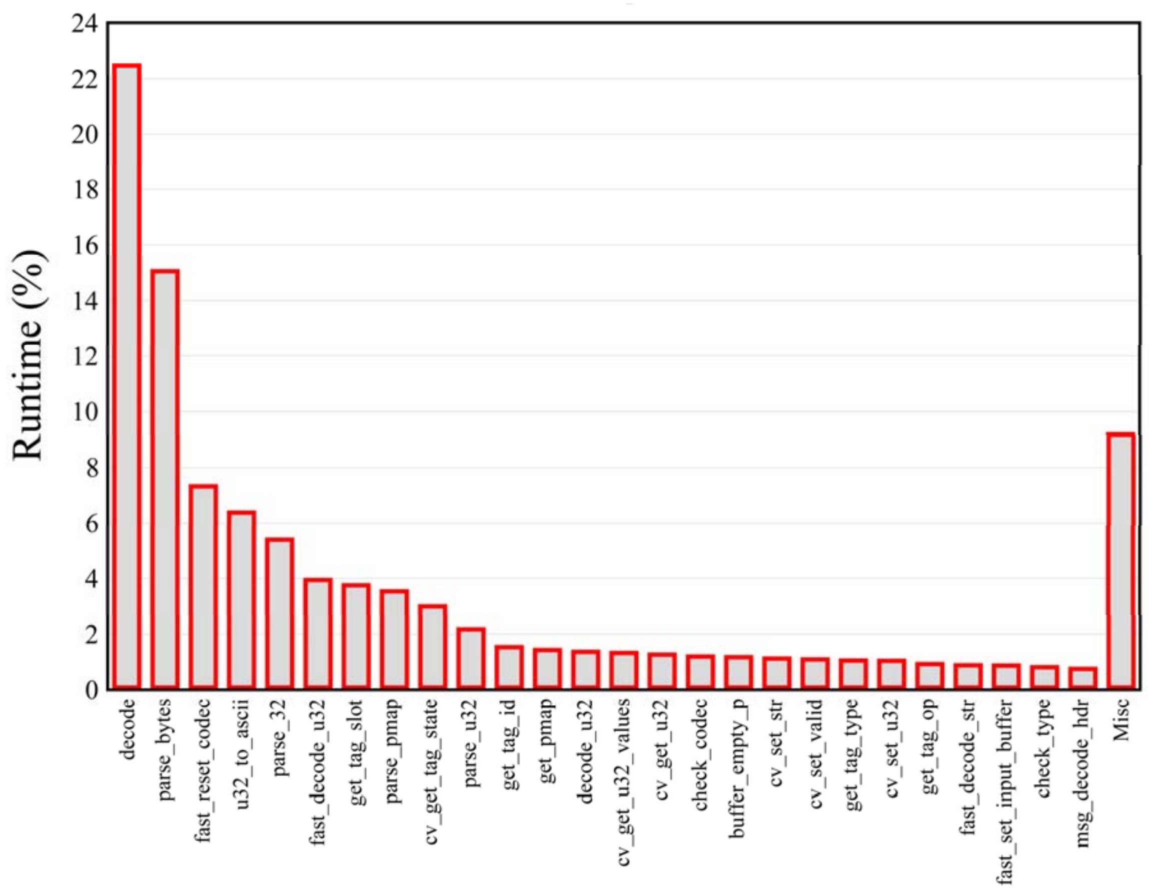
OPRA : Reference decoder



OPRA : Reference decoder profiling

Complex control flow structure:

execution time distributed across all kernels.



OPRA : Contribution

1. High-speed hand-optimized OPRA decoder implemented using bottom-up approach (using highly optimized building blocks)

2. Using **DSParser** to capture the essence of OPRA protocol in a handful of high level description language.

3. Extensive performance analysis and comparison to analyze the optimality of our DSParser approach over hand-optimized kernels.

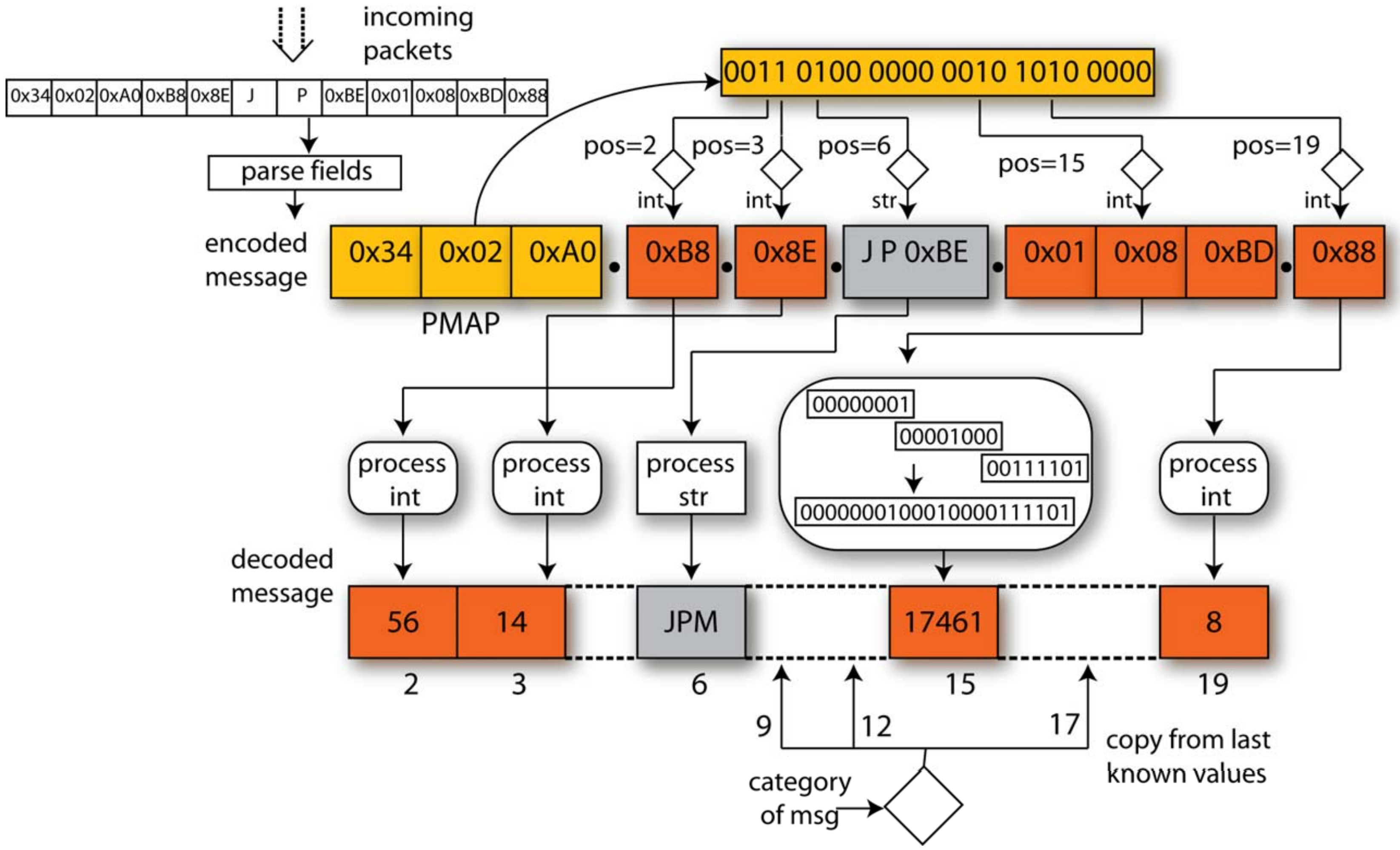
OPRA : Bottom-up approach

- Identify compute intensive kernels and optimize them independently
- High performance through assembly level optimization
- Hand made input data tokenization
- Detailed CPU architecture analysis to understand available hardware mechanisms.

OPRA : Optimized kernels

- **Category specific bitmaps**
  - Every message type requires a subset of fields (from the total specification).
  - Reference decoder has complex control structure due to different actions for every type.
  - We introduce category specific bitmaps that specify the fields relevant to every message type.
    - An AND of PMAP with BMAP specifies the new fields present.
    - An XOR of PMAP with BMAP specifies ones that need to be copied.
- **PMAP parsing**
  - Reference decoder checks every bit of PMAP.
  - We use count leading zeroes (clz) to get next field index.
- **Integer conversion**
  - Reference decoder proceeds one byte at the time.
  - We use a different optimized code for each integer length.

OPRA : Decoding Algorithm



DSParser rationale

- Tokenization/parser application can be expressed in terms of regular expression fragments that match protocol components
- These fragments must be composed in such a way to describe the actual communication protocol
- We designed an high level domain specific language that describes how regular expression fragments are composed and how “interesting” data can be extracted from the stream
- We used imperative language like syntax

OPRA : DSParser control flow

```
MATCH "........................................"
MATCH "\x01\x02"
PUSH
MATCH ".........."
EXECUTE sequence_number
MATCH "..."
LOOP
    WHILECNT "."
        PUSH
        MATCH "[\x00-\x7f]*[\x80-\xff]"
        EXECUTE action_pmap
        PUSH
        WSWITCH
            CASE "[\x80-\xff]"
                EXECUTE action_field
                PUSH
            ENDCASE
            CASE "[\x00-\x7f]"
                MATCH "[\x00-\x7f]*[\x80-\xff]"
                SEND 0x02 0
                EXECUTE action_field
            ENDCASE
        ENDWSWITCH
    ENDWHILECNT
ENDLOOP
```

Uses optimized kernels from bottom up approach for OPRA field actions.

DSParser approach

The parser generator leverages the regular expression capabilities of our DotStar regular expression engine:

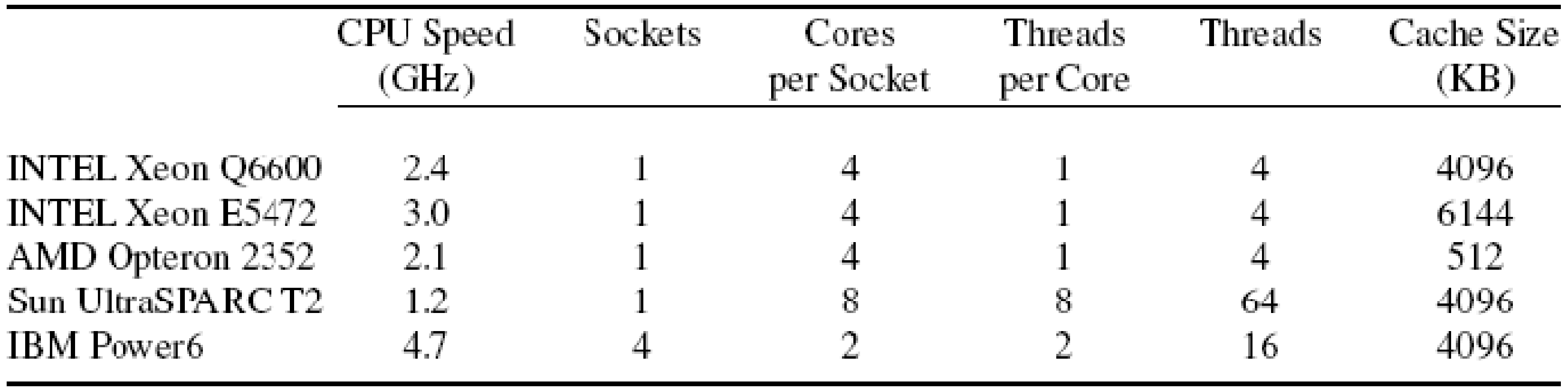
- Individual regexp fragments are compiled into a Deterministic Automaton containing direct and “failure” transitions
- High level constructs are represented as data flow graphs that contain a subgraph and have a failure transition
- Regular expression fragment DFA are inserted inside the data flow graph and remaining failure transitions are computed

DSParser runtime

Is a very simple runtime, executing over a small FSM, and has several interesting properties:

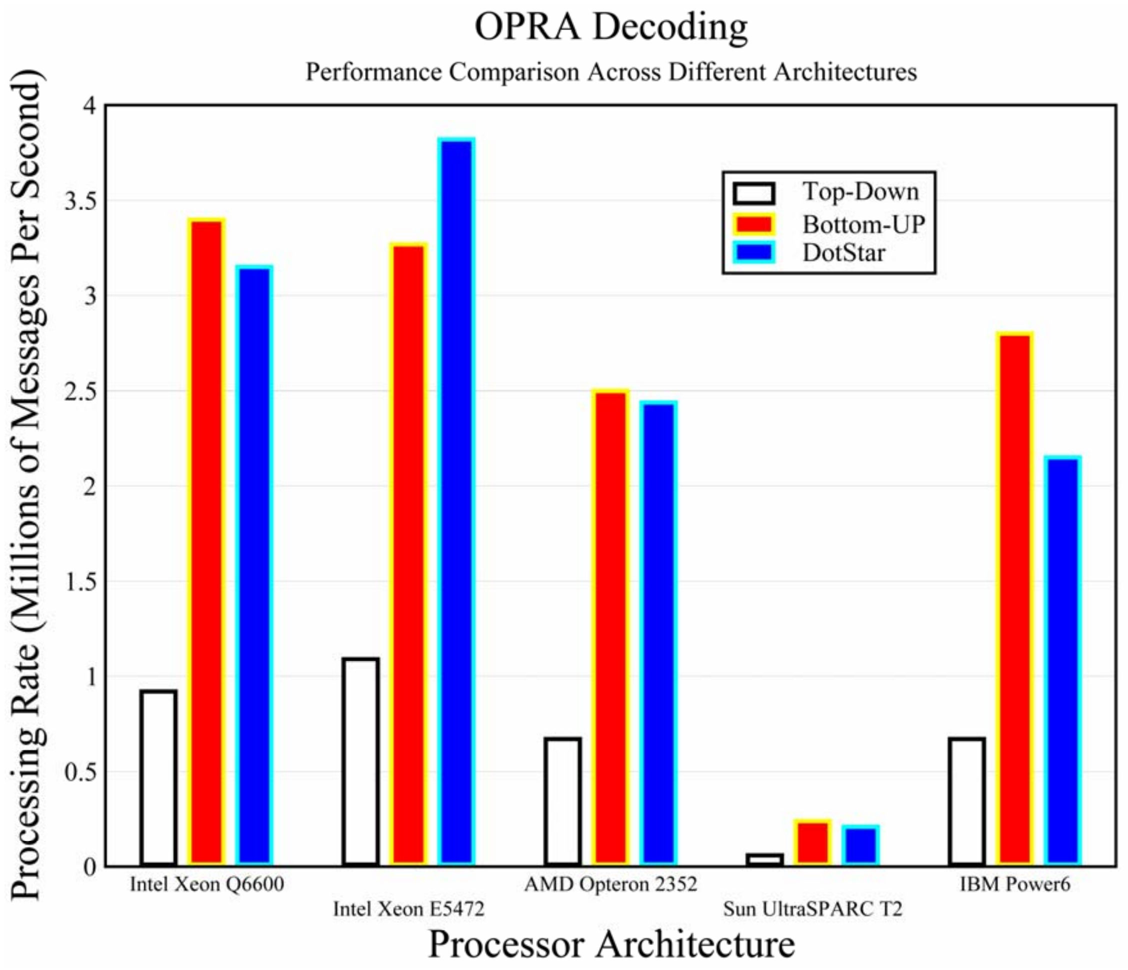
- It uses little memory and normally fits entirely in cache
- It does not require any specific vector extension or special instruction
- It leverages pipelined memory access for accessing FSM data structures and input data
- Branch-less

OPRA decoding: Performance Analysis

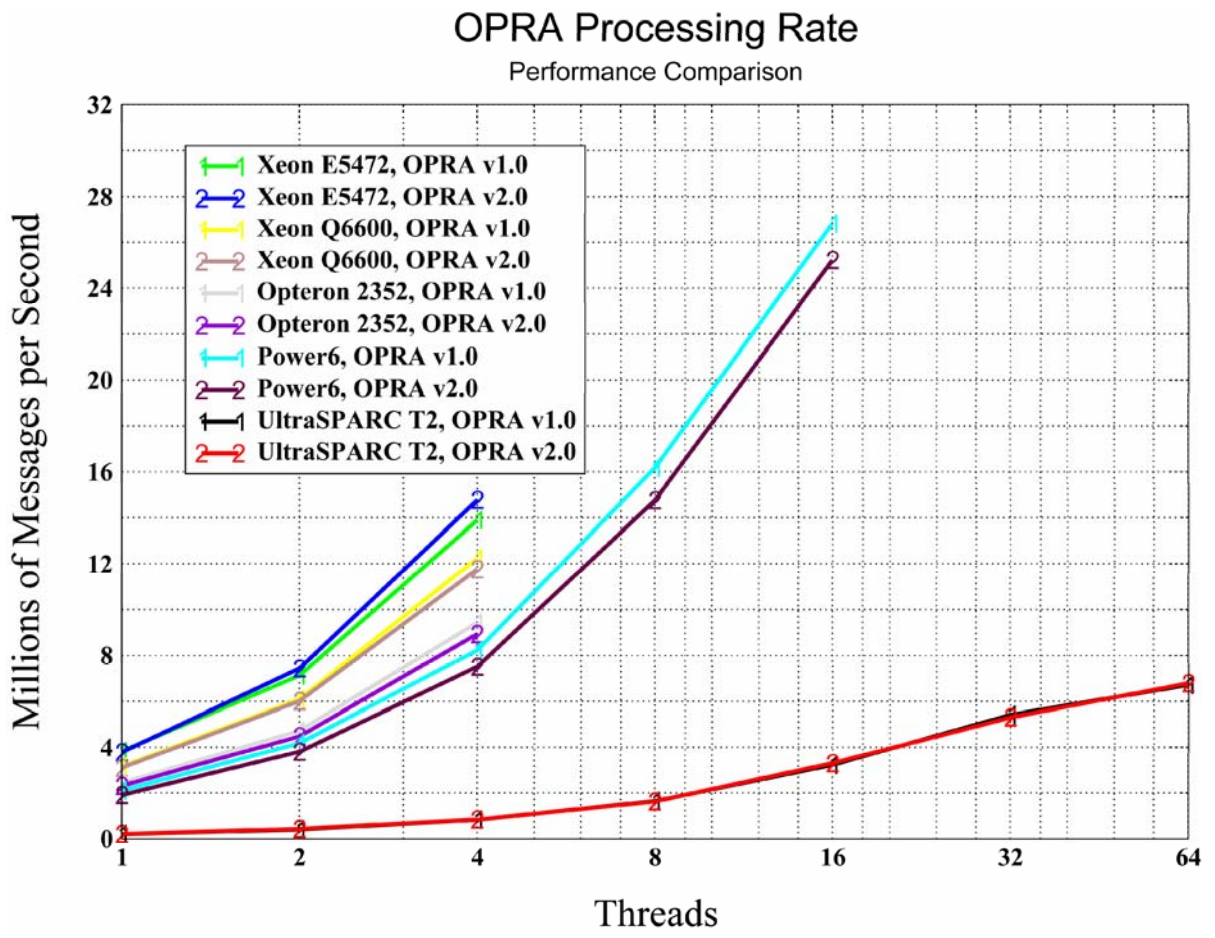


	CPU Speed (GHz)	Sockets	Cores per Socket	Threads per Core	Threads	Cache Size (KB)
INTEL Xeon Q6600	2.4	1	4	1	4	4096
INTEL Xeon E5472	3.0	1	4	1	4	6144
AMD Opteron 2352	2.1	1	4	1	4	512
Sun UltraSPARC T2	1.2	1	8	8	64	4096
IBM Power6	4.7	4	2	2	16	4096

OPRA decoding: Performance Analysis



OPRA decoding: Performance Analysis



Conclusion

- High Performance Computing is expanding from scientific community towards financial market daily operations
- Modern CPU architecture implement several “lesson learned” in larger previous HPC systems
- Financial institution requirements can be fulfilled only by using HPC techniques and technologies on modern commodity hardware
- Latest generation multicore CPU are better performant than corresponding FPGA implementation