Parallel Computing Strategies and Implications

Multi-process vs. multi-threaded

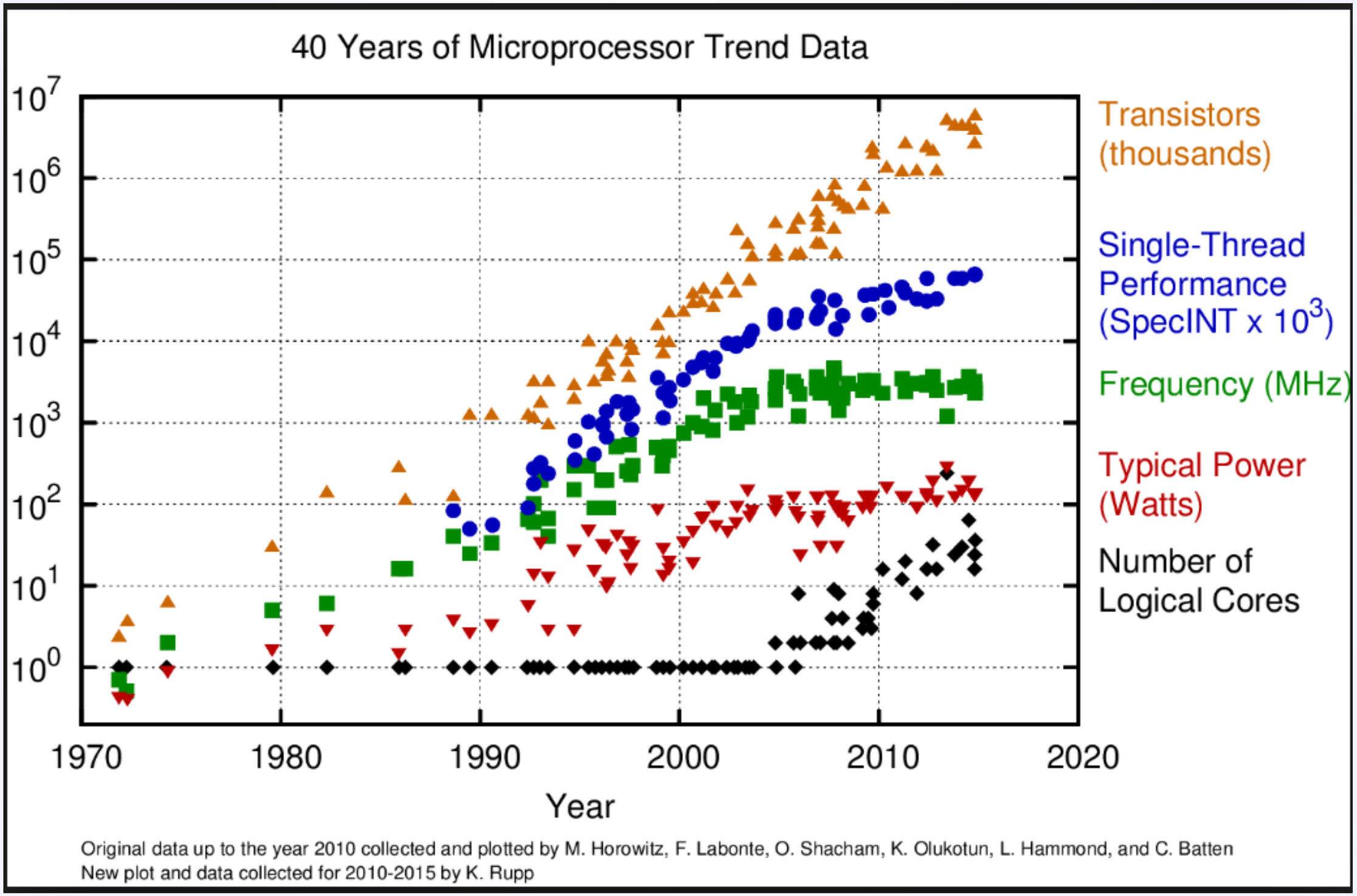
IncrediBuild
BEYOND ACCELERATION

Dori Exterman
CTO IncrediBuild

In this session we will discuss…

- Multi-threaded vs. Multi-Process
- Considerations for making your choice
- Review common market use cases
- Compare a specific use-case
- Scalability
- Summary and Q&A

Moore's Law Going Multi-Core



Moore's Law Going Multi-Core

Increase in clock speed, allowing software to automatically run faster, **no longer oblige** to Moore's law

Chip manufacturers keep Moore's law by **increasing the number of CPUs**

Software has to be written to take advantage of all the **multi-processors**.

Instead of doing everything in a linear fashion, programmers need to **ensure applications are designed for parallel tasking**.

Choosing the Right Parallel Compute Architecture is Important

- Time to market
- Future product directions
- Product's maintainability
- Product's complexity

Multi-threaded: Pros

Easy and fast to communicate **and share data between threads** (especially read-only data).

Easy to **communicate with parent process**.

Beneficial for **large datasets** that can't be divided to sub sets.

Supported by many **libraries**.

Multi-threaded: Cons

- One thread **crashes** → the entire process crashes
- Multi-threaded apps are hard to **debug**
- Writing to shared data requires (difficult to maintain) **locks**
- Too many threads → too much time in **context-switching**.
- **Not scalable** to other machines / public cloud

Multi-process : Pros

- Includes **multi-threaded by definition.**
- Process **crash won't harm** other processes.
- Much **easier to debug**
- **Encapsulated** – easier to manage for large teams
- **Scalable** – distributed computing \ public cloud
- More **memory** can be consumed

Multi-process: Cons

Custom development required in order to **communicate between processes**

Requires special mechanism to **share memory data between processes**

Limited “process-safe” libraries

Multi-threaded vs. Multi-core: List of Considerations

- Parallel parts need to sync data?
- Data sets size
- Ease of development
- Scaling - Compute time and throughput

Synchronization/State Maintenance

Easier in multi-threaded

Easier to debug in multi-threaded

In multi-process, will require custom development

Many mechanisms exists both for multi-process and multi-threaded

Ease of Development

Pre-made **libraries for MT** (multi-threaded)

MP - **Easier debug/maintenance/code understanding** / **less bugs** will be introduced with new commits

MP - Can be developed and maintained by **less experienced developers**

MP - Requires developer to maintain **encapsulation**

MP - **Bugs are less complicated to solve** when MT is not involved

Throughput & scalability - 1/2

How many tasks do I have to execute in parallel – dozens /millions

How many tasks will my product need to support in the future?

How much time will each task take? Milliseconds or minutes?

How much memory will each task require?

How large is the data that each task requires?

Throughput & scalability - 2/2

- How complex is each task's code?
- Is my system real-time?
- Would the product benefit from scalability?
- Business wise - scalability as a service?
- Offloading

Other Scalability Questions

Do my tasks require special hardware?

Will my tasks perform better with specific hardware?

Connection to a database

Common scenarios for using multi-threaded

- Simple scenarios
- Real-time performance
- Long initialization time with short computation time
- Communication, synchs and states
- Only a few highly intensive tasks

Common scenarios for using multi-process

- Multiple, encapsulated modules
- Tasks may require much memory
- Limited synchronization required

- Large application \ large team
- Scaling \ High-throughput
- Tasks are complex and prawned to errors

Parallel Computing

The **High-Throughput** Challenge

Let's have a look at a real-life scenario

Maven vs **Make** build tools

- Popular tools
- Many tasks
- Tasks are atomic
- Minimal communication
- Small dataset
- Scalable

Scenarios – Build system approach: Maven

- Multi-threaded code was only added recently:

Confluence Spaces

Apache Maven

Pages

Blog

Space tools

Parallel builds in Maven 3

Pages / Index / Proposals

Parallel builds in Maven 3

1 Added by Kristian Rosenvold, last edited by Jason van Zyl on Dec 06, 2013 (view change) show comment

Maven 3.x has the capability to perform parallel builds. The command is as follows:

mvn -T 4 clean install # Builds with 4 threads
mvn -T 1C clean install # 1 thread per cpu core
mvn -T 1.5C clean install # 1.5 thread per cpu core

This build-mode analyzes your project's dependency graph and schedules modules that can be built

Scenarios – build system approach: Maven

- Not Scalable: Users are asking for it, but it is still missing multi-process invocation.
- Projects are building many 3rd parties.

Distributed build with maven?

Launch yourself. stackoverflowcareers

0

Currently, we have a maven project with a few thousand tests that takes two hours to run. We've tried running these in parallel, but since they are functional tests, each test configures the system in a specific way, which leads to race conditions and random test failures. What I'd like to do is spin up N servers on AWS, then have maven divide my tests up and run them on these servers (each

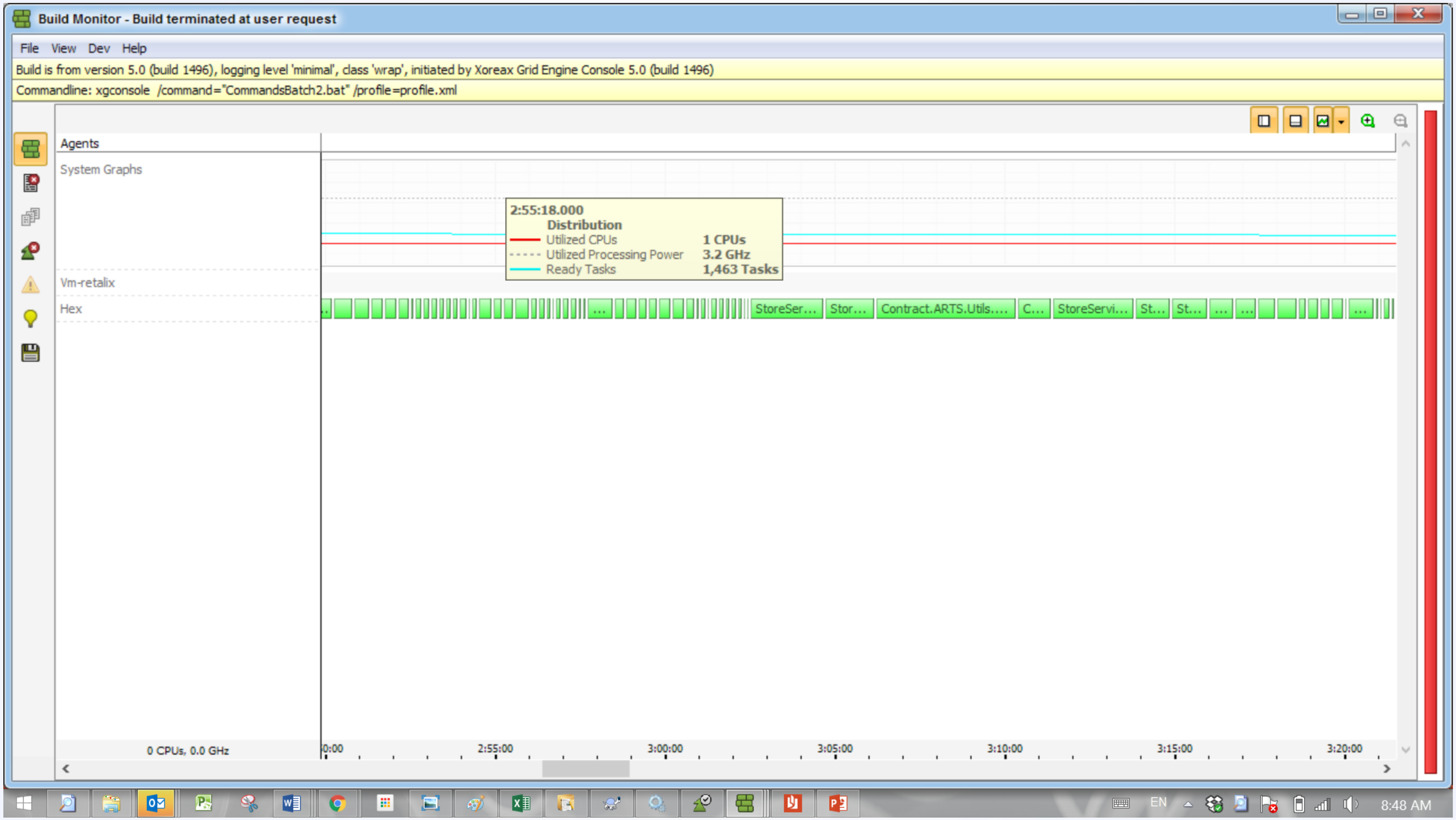
Launch yourself. stackoverflowcareers

3

Distcc makes it easy to distribute a C or C++ compile job across a number of machines, and is a godsend for working with large, frequently-built codebases.

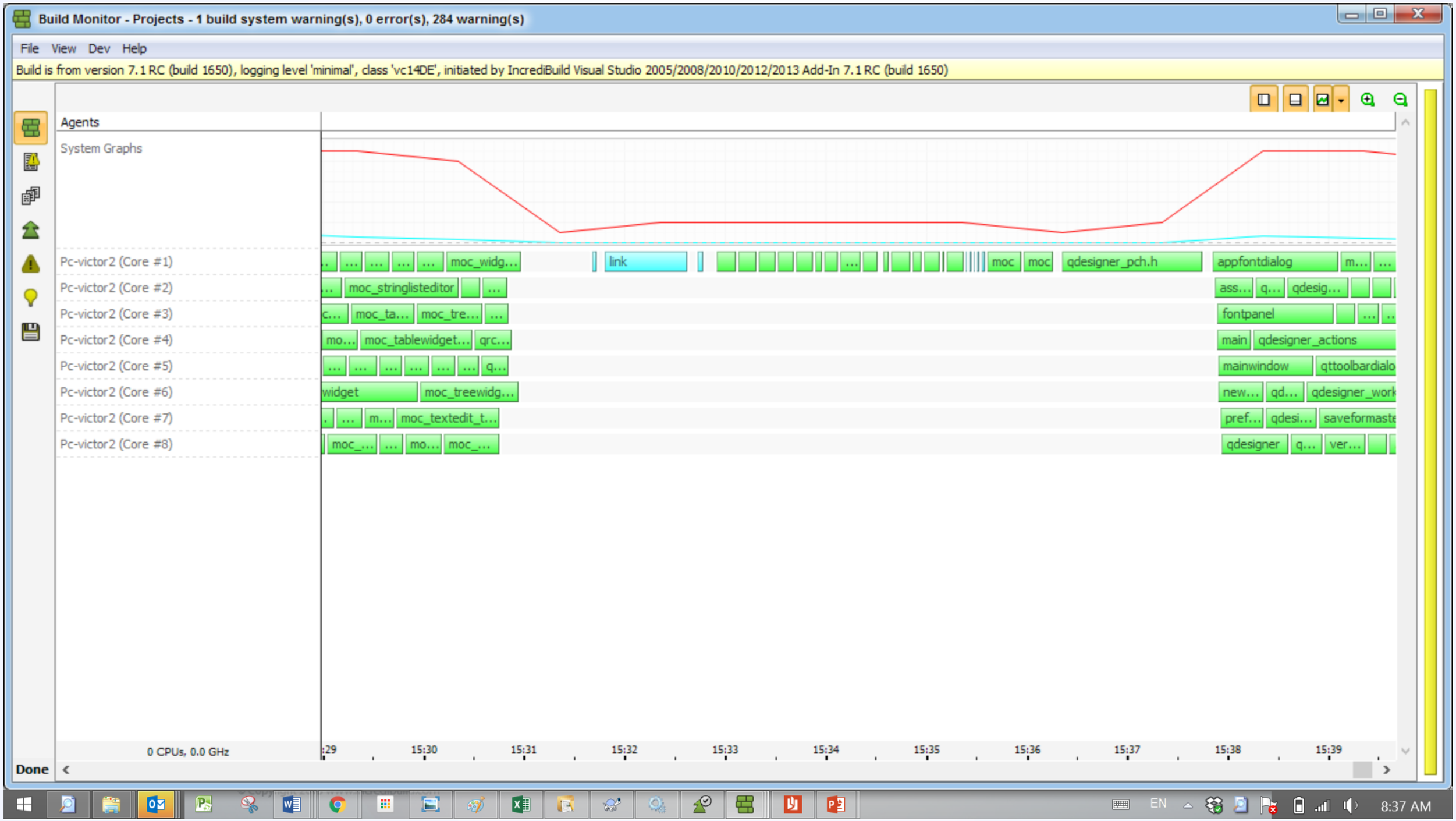
An increasing number of our large projects are built in Java these days, however. Does anyone know of something equivalent or similar for Java? While it would be great if it would go down to the **javac** level, our multi-module projects would benefit from being able to distribute each maven 2

Single Machine, Single Core



Single Machine – Multi-Core

8 cores, 15:45 minutes

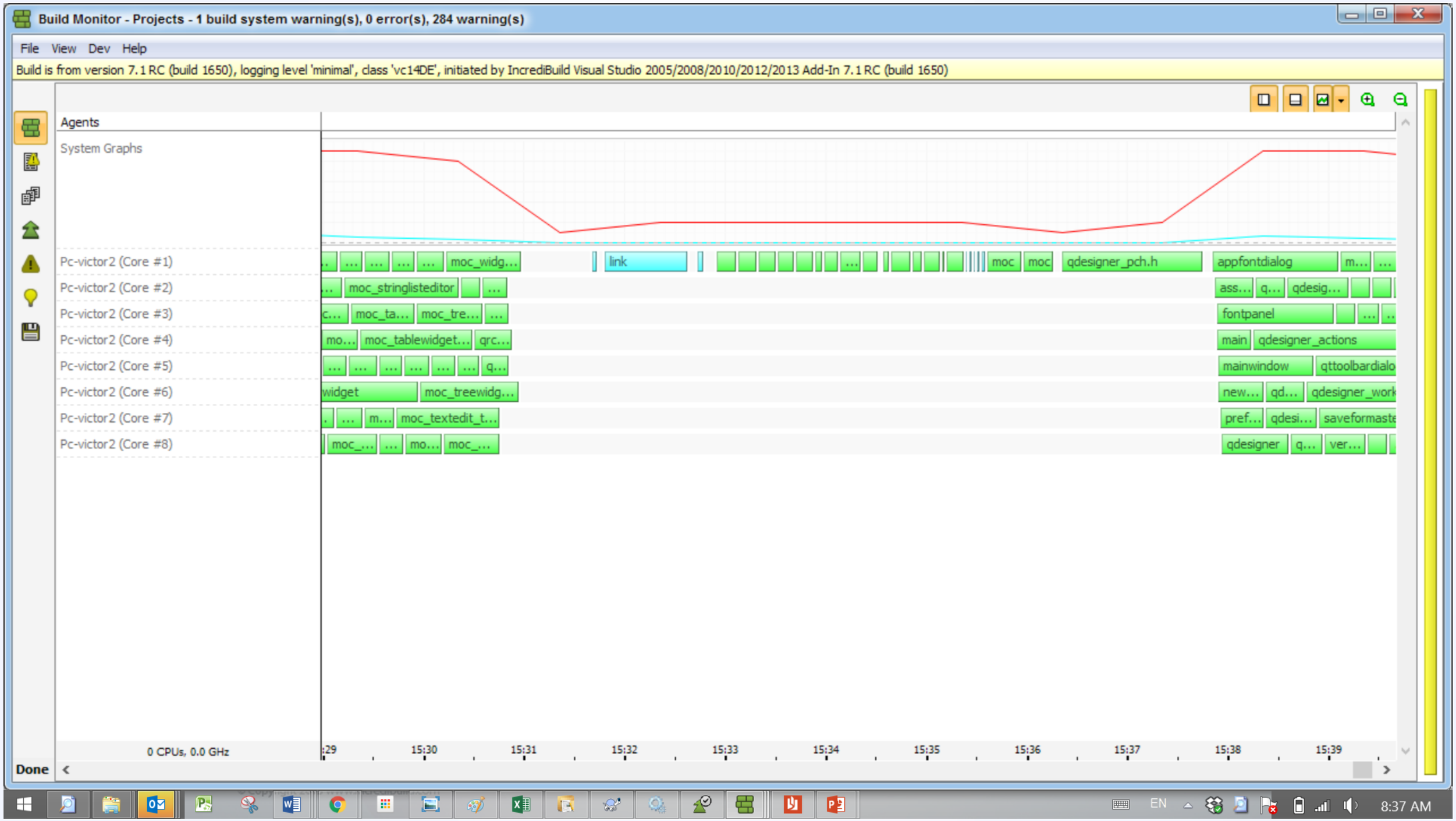


Make

- Multi-process (same as other modern build tools)
- Better memory usage
- Caching add-ons (CCache)
- Scalable – 3rd party solutions (DistCC, IceCC, IncrediBuild)

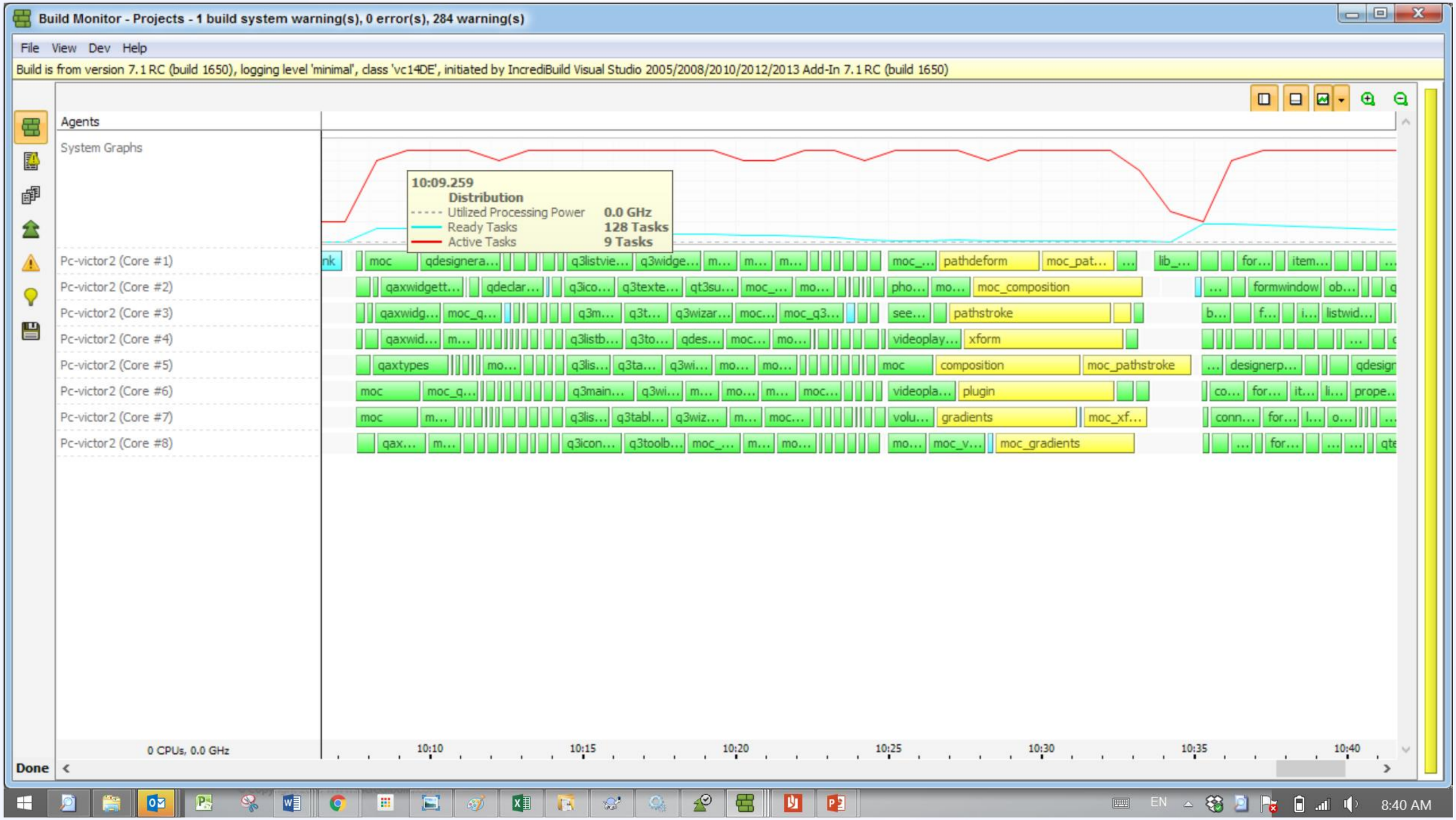
QT on Single Machine – Multi-Core

8 cores, 15:45 minutes



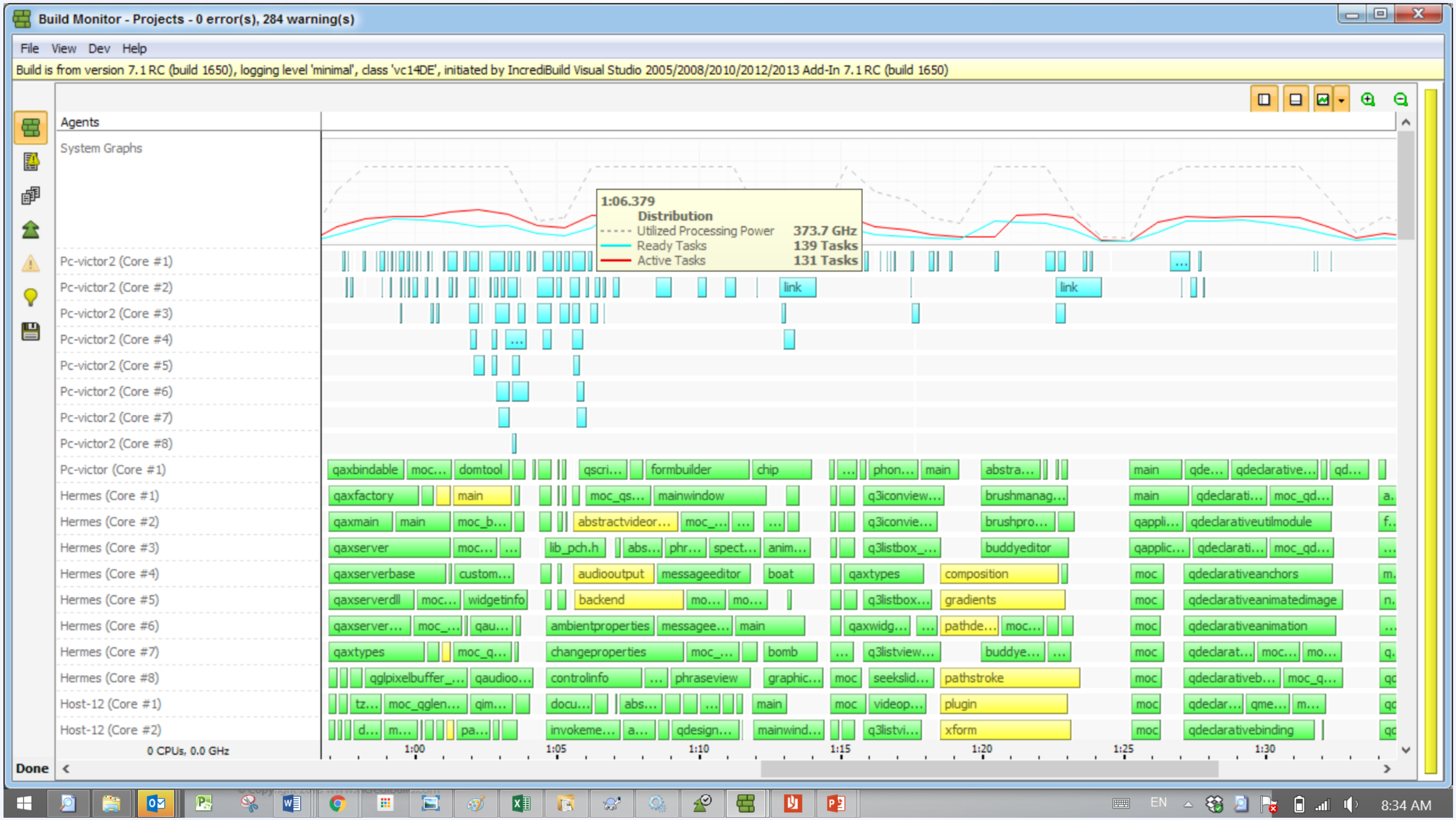
QT on Single Machine – Multi-Core (predicted)

8 cores, 11 minutes



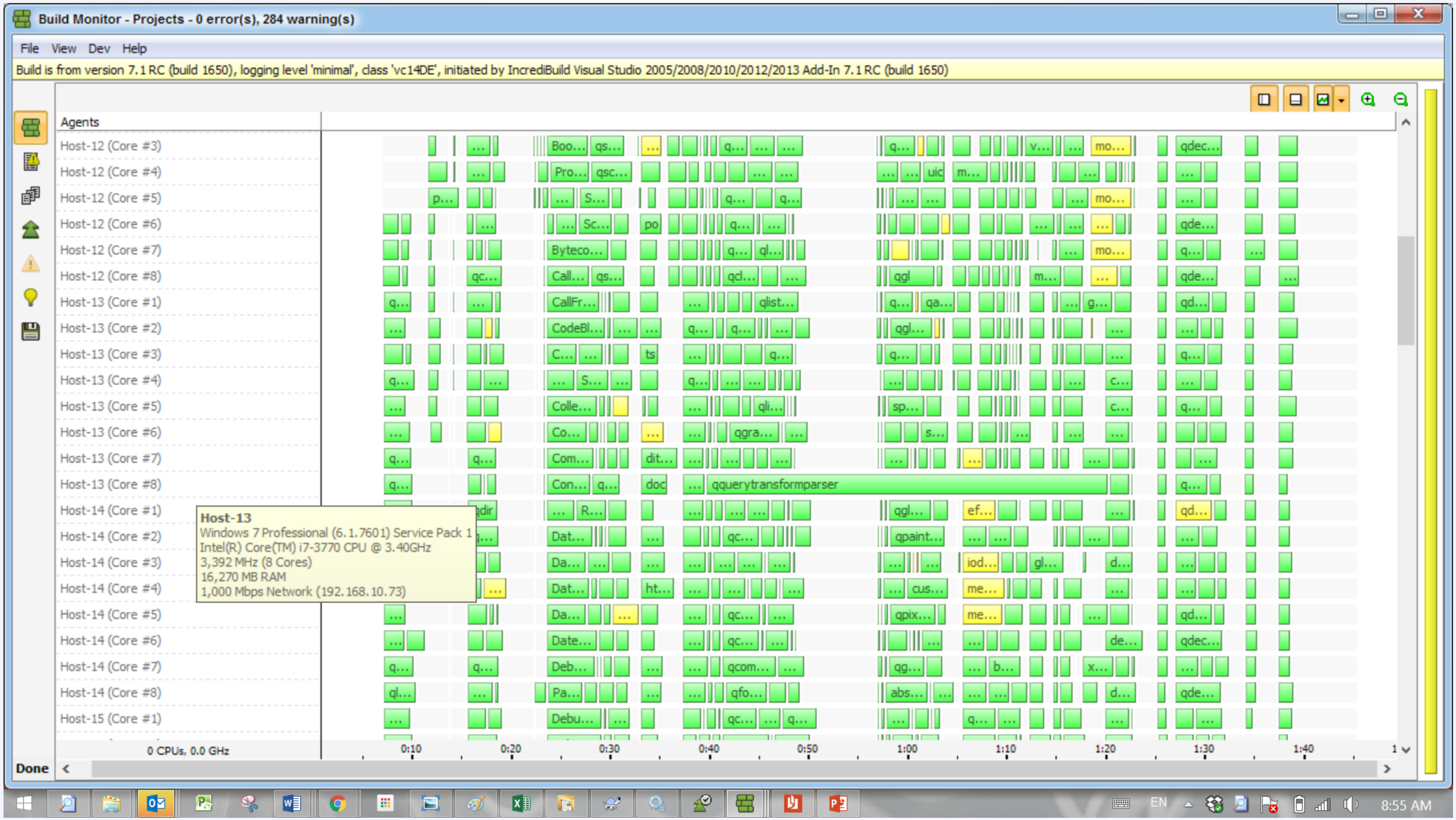
QT on multiple machines, ~200 cores

130 cores, 1:40 minutes



QT on multiple machines, multi-core

130 cores, 1:40 minutes



Real Life Scenarios

Common Multi-Threaded Scenarios

Common high-throughput scenarios and decision

Large in-memory read \ write meshes:

- CAD
- AI (connections between elements)
- Weather forecasting
- Genetic algorithms

Needs all the **data to be in-memory and accessible** for all tasks.

Common high-throughput scenarios and decision

Real time performance needs:

- Financial transactions
- Defense systems
- Gaming

Latency and initialization time are very important.

Common high-throughput scenarios and decision

Applications which most of the calculation (business logic) is **Database related** (stored procedures, triggers, OLAP)

CRM, ERP, SAP

Common high-throughput scenarios and decision

Fast sequential processing:

- CnC
- Manufacturing line
- Dependent calculations

Next calculation depends on the previous one.

Common Multi-Process Scenarios

Common high-throughput scenarios and decision

Large **independent dataset**: when we don't need all data in-memory at the same time.
Low dependencies between data entities.

- Simulations (transient analysis) – Ansys, PSCad
- Financial derivatives – Morgan Stanley
- Rendering

Common high-throughput scenarios and decision

Many calculations with small business units.
Solution: batch tasks

- Compilations
- Testing (unit tests, API tests, stress tests)
- Code analysis
- Asset builds
- Math calculations (Banking, Insurance, Diamond industry).

Common high-throughput scenarios and decision

GPU – applications that scale to use many GPUs

- CUDA, OpenCL
- Rendering
- Password cracking

Common high-throughput scenarios and decision

Output and computation streaming:

NVIDIA Shield, Onlive Service, Netflix, Waze

The business aspect

- Scalability
- Offloading
- Full cloud solution
- SAAS

Summary

Summary - Writing Scalable Apps:

Architectural considerations:

- Can my execution be broken to many independent tasks?
- Can the work be divided over more than a single host?
- Do I have large datasets I need to work with?
- How many relations are in my dataset?
- How much time would it take my customers to execute a large scenario – do I have large scenarios?

Summary - Writing Scalable app

Technical considerations:

- Do I have any special communication and synchronization requirements?

- Dev Complexity - What risks can we have in terms of race-conditions, timing issues, sharing violations – does it justify multi-threading programming?

- Would I like to scale processing performance by using the cloud or cluster?

Distributed Computing

Some of the known infrastructures to using multi-core/multi-machine solutions:

- MapReduce (Google – dedicated grid)
- Hadoop (Open source – dedicated grid)
- Univa Grid Engine(Propriety – dedicated grid)
- IncrediBuild (Propriety - ad-hoc \ process virtualization grid)

Q&A

Dori Exterman, IncrediBuild CTO
dori@incredibuild.com

IncrediBuild now free in Visual Studio 2017
www.incredibuild.com

Thank you.