Coding the Continuum

Ian Foster

Argonne Training Program on Extreme-Scale Computing
August 7, 2019

Supercomputing

1.6 x 10^8 flop/s (Cray-1)  1.5 x 10^{17} flop/s (Summit)

**Supercomputing**

$1.6 \times 10^8$ flop/s (Cray-1)  $1.5 \times 10^{17}$ flop/s (Summit)

**Wide area networking**

$10^4$ bit/s (dial-up)  $10^{11}$ bit/s (optical)
"When the network is as fast as the computer’s internal links, the machine disintegrates across the net into a set of special-purpose appliances."

-- George Gilder, 2000
"When the network is as fast as the computer’s internal links, the machine disintegrates across the net into a set of special-purpose appliances."

-- George Gilder, 2000
“network is as fast as the computer’s internal links”

Communication technologies continue to evolve

Hollow core fiber:
- 99.7% speed of light
- (1.46x faster than fiber)
- 73.7 terabits per second

5G is transforming communications

Global IP traffic, wired and wireless

5G CAGR 2017-2022

Innovation continues in the lab

doi.org/10.1007/978-3-319-31903-2_8

doi:10.1038/nphoton.2013.45
We can compute anywhere!

- Cheapest
- Nearest to data
- Greenest
But are we really free?

Time = $T_{\text{compute}} + 2 \ T_{\text{latency}}$

300 000 km per second*. It’s not just a good idea... *It’s the law*

*207 000 km/sec in optical fiber: $5 \times 10^{-6}$ sec/km

Uphill in all directions
"When the network is as fast as the computer’s internal links, the machine disintegrates across the net into a set of special-purpose appliances."

-- George Gilder, 2001
"a set of special-purpose appliances"

<table>
<thead>
<tr>
<th></th>
<th>More Flexible</th>
<th>Perf/W</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>More Flexible</td>
<td>1X</td>
<td>Today's standard, most programmable, good for services changing rapidly</td>
</tr>
<tr>
<td>Manycore CPUs</td>
<td>Homogeneous</td>
<td>3X</td>
<td>Many simple cores (10s to 100s per chip), useful if software can be fine-grain parallel, difficult to maintain.</td>
</tr>
<tr>
<td>GPUs</td>
<td>Specialized</td>
<td>5-30X</td>
<td>Good for data parallelism by merged threads (SIMD), High memory bandwidth, power hungry</td>
</tr>
<tr>
<td>FPGAs</td>
<td>Specialized</td>
<td>5-30X</td>
<td>Most radical fully programmable option. Good for streaming/irregular parallelism. Power efficient but currently need to program in H/W languages.</td>
</tr>
<tr>
<td>Structured ASICS</td>
<td>Specialized</td>
<td>20-100X</td>
<td>Lower-NRE ASICs with lower performance/efficiency. Includes domain-specific (programmable) accelerators.</td>
</tr>
<tr>
<td>Custom ASICs</td>
<td>Specialized</td>
<td>&gt; 100X</td>
<td>Highest efficiency. Highest NRE costs. Requires high volume. Good for functions in very widespread use that are stable for many years.</td>
</tr>
</tbody>
</table>

Source: http://bit.ly/2SDGHzT
### AI chip landscape

**Tech Giants/Systems**
- Google
- Microsoft
- AWS
- IBM
- Facebook
- Apple
- Tesla
- Huawei
- Alibaba Group
- Fujitsu
- NOKIA
- Toshiba
- Hewlett Packard Enterprise
- Rockchip

**IC Vender/Fabless**
- Intel
- Samsung
- NVIDIA
- Qualcomm
- AMD
- Xilinx
- NXP
- STMicroelectronics
- MediaTek
- Fujitsu
- NOKIA
- Toshiba

**IP/Design Service**
- Arm
- Synopsys
- Imagination
- Cadence
- CEVA
- VeriSilicon
- SiFive

**Startup in China**
- Cambricon
- Nanjing Horizont Robotics
- Bitmain
- Intelusion
- ChipIntelli
- ThinkForce
- Canaan
- Rokid
- Nuvia
- Enflame
- 亿智科技

**Startup Worldwide**
- Cerebras
- WAVE Computing
- Graphcore
- Habana
- Thinc
- SambaNova
- Kalray
- Lightelligence
- HaiLO
- Esperanto Technologies
- Preferred Networks
- Brainchip
- PEZY Computing
- GreenWaves Technologies
- Mythic
- AMOTIVE
- Konik
- Tachyum

**Compiler**
- XDA
- Glow
- LLVM
- NVIDIA TensorRT
- nGraph Compiler stack (Beta)

**Benchmarks**
- MLPerf
- AI - Benchmark
- AI Matrix
- AWA
“a set of special-purpose appliances”

“Cloud computing 5x to 10x improved price point [relative to Enterprise]”

Why?
• Improved utilization
• Economies of scale in operations
• More power efficient
• Optimized software
Google hyperscale data center, St. Ghislain, Belgium
Zero-carbon cloud: Reduce energy cost and energy carbon footprint to 0

Andrew Chien  DOI 10.1109/IPDPS.2016.96
The performance landscape becomes peculiar

A program can run on two computers
  On $C_1$, it takes 0.01 seconds
  On $C_2$, it takes 0.005 seconds
Which is faster?
The performance landscape becomes peculiar

A program can run on two computers
  On $C_1$, it takes 0.01 seconds
  On $C_2$, it takes 0.005 seconds
Which is faster?

The answer depends on their location.
Say $C_1$ is adjacent and $C_2$ is 500 km distant
  $t(C_1) = T_1 = 0.01 \text{ sec}$
  $t(C_2) = T_2 + 2 \times 500 \times 5 \times 10^{-6} = 0.01 \text{ sec}$
The performance landscape becomes peculiar

A program can run on two computers
   On $C_1$, it takes 0.01 seconds
   On $C_2$, it takes 0.005 seconds
Which is faster?

The answer depends on their location.
Say $C_1$ is adjacent and $C_2$ is 500 km distant
   $t(C_1) = T_1 = 0.01$ sec
   $t(C_2) = T_2 + 2 \times 500 \times 5 \times 10^{-6} = 0.01$ sec

The apparent **speed** of a computer depends on its **location**;
the apparent **location** of a computer depends on its **speed**
Continuum
A set of elements such that between any two of them there is a third element [dictionary.com]

For example, the computing continuum:

<table>
<thead>
<tr>
<th>IoT/Edge</th>
<th>Fog</th>
<th>HPC/Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Nano</td>
<td>Micro</td>
</tr>
<tr>
<td>Example</td>
<td>Adafruit Trinket</td>
<td>Particle.io Boron</td>
</tr>
<tr>
<td>Memory</td>
<td>0.5K</td>
<td>256K</td>
</tr>
<tr>
<td>Network</td>
<td>BLE</td>
<td>WiFi/LTE</td>
</tr>
<tr>
<td>Cost</td>
<td>$5</td>
<td>$30</td>
</tr>
</tbody>
</table>

Count = $10^9$
Size = $10^1$

Credit: Pete Beckman, beckman@anl.gov
The space-time continuum

“space by itself, and time by itself, are doomed to fade away into mere shadows, and only a kind of union of the two will preserve an independent reality ...”

H. Minkowski, 1908

Space-time diagram
https://en.wikipedia.org/wiki/Spacetime
The spacetime continuum in computational systems

The behaviors of the two computers are indistinguishable

\[ t(C_1) = T_1 = 0.01 \text{ sec} \]
\[ t(C_2) = T_2 + 2 \times 500 \times 5 \times 10^{-6} = 0.01 \text{ sec} \]
The spacetime continuum in computational systems

Misquoting Minkowski: “Henceforth, location for itself, and speed for itself shall completely reduce to a mere shadow, and only some sort of union of the two shall preserve independence.”

The behaviors of the two computers are indistinguishable

t(C₁) = T₁ = 0.01 sec

t(C₂) = T₂ + 2 \times 500 \times 5 \times 10^{-6} = 0.01 sec
A real example: High energy physics trigger analysis

Local: 2000 msec
Remote: 30 + 10 + 10 = 50 msec

40x acceleration

T₁ = 2 seconds on CPU (not to scale)
T₂ = 30 msec on FPGA

2000 km (Virginia)
0 km (Illinois)

Speed of light →10 ms

Reasoning about the computing continuum (a) Assumptions

A1: N consumers, distributed uniformly, X secs apart

A2: Each consumer requests compute units at 1 Hz

A3: Infinite bandwidth: i.e., only compute time and latency are of concern

A4: An individual computer takes T secs to complete a compute unit

A5: A compute center containing Z computers is faster by a factor of $\sqrt{Z}$
Reasoning about the computing continuum
(a) Assumptions

**A1**: N consumers, distributed uniformly, X secs apart

**A2**: Each consumer requests compute units at 1 Hz

**A3**: Infinite bandwidth: i.e., only compute time and latency are of concern

**A4**: An individual computer takes T secs to complete a compute unit

**A5**: A compute center containing Z computers is faster by a factor of $\sqrt{Z}$

* Grosch's law (1953): computer power rises by the square of the price
Reasoning about the computing continuum (b) Without response time bounds

Max time is:

A) 1 local computer per consumer: 
   \[ t = T \]

B) Center with N computers: 
   \[ t = \frac{T}{\sqrt{N}} + 2 \sqrt{\frac{N}{2}} X \]
Reasoning about the computing continuum
(b) Without response time bounds

Max time is:

E.g., $N = 100$, $T=0.01$, $X=0.0001$

A) 1 local computer per consumer:

\[ t = \frac{T}{N} = 0.01 \text{ s} \]

B) Center with $N$ computers:

\[ t = \frac{T}{\sqrt{N}} + 2 \sqrt{\frac{N}{2}} X \]

\[ = \frac{0.01}{10} + 2 \sqrt{\frac{100}{2}} 0.0001 \]

\[ = 0.001 + 0.00142 = 0.00242 \text{ s} \]
Reasoning about the computing continuum (b) Without response time bounds

Max time is:

E.g., \(N = 100, \ T=0.01, \ X=0.0001\)

A) 1 local computer per consumer:

\[ t = T = 0.01 \text{ s} \]

B) Center with \(N\) computers:

\[ t = \frac{T}{\sqrt{N}} + 2 \sqrt{\frac{N}{2}} \ X \]

\[ = \frac{0.01}{10} + 2 \sqrt{\frac{100}{2}} \cdot 0.0001 \]

\[ = 0.001 + 0.00142 = 0.00242 \text{ s} \]

Centralized scheme is 4x faster
Reasoning about the computing continuum (c) With response time bound, B

We want to know D for which:
\[
\frac{T}{\sqrt{\text{size}}} + 2D \leq B
\]
As size is \(\pi D^2/X^2\), we want to solve:
\[
\frac{T}{\sqrt{\pi D^2/X^2}} + 2D = B
\]

From A1, there are \(\pi D^2/X^2\) consumers within distance D of a compute center.
Reasoning about the computing continuum (c) With response time bound, B

We want to know D for which:
$$\frac{T}{\sqrt{\text{size}}} + 2D \leq B$$

As size is $\pi D^2/X^2$, we want to solve:
$$\frac{T}{\sqrt{\pi D^2/X^2}} + 2D = B$$

With $B=0.01$, $T=0.001$, $X=0.0001$ sec:
$$D = 0.004964 \text{ sec (~1000 km)}$$

Then:
- Size = $\pi D^2/X^2 = 7854$
- Max processing time is
  $$2 \times 0.004964 + 0.001/\sqrt{7854} = 0.01 \text{ seconds}$$

From A1, there are $\pi D^2/X^2$ consumers within distance D of a compute center.
Reasoning about the computing continuum
(d) Discussion

The model emphasizes the importance of distribution ("disintegration") of function and aggregation of capability

The model can be improved:
• Empirical data on scaling of cost and speed with size
• Data transfer costs
• Empirical data on workloads

Optimal solutions will involve compute centers of multiple sizes
→ Not just "center" and "edge"
Small and midsize data centers: Server intensity

Source: LBNL-2001025
Coding the continuum

Code: verb.

1) to arrange or enter in a code
Coding the continuum

Code: verb.

1) to arrange or enter in a code
2) to write code for
Coding the continuum

Code: verb.
1) to arrange or enter in a code
2) to write code for

Now that the machine has disintegrated across the net, how do we program it?
Coding the continuum

Code: verb.
1) to arrange or enter in a code
2) to write code for

Now that the machine has disintegrated across the net, how do we program it?
An example: Serial synchrotron crystallography
An example: Serial synchrotron crystallography

For each sample:
• Image crystals at ~50 Hz:
  • Validate each image
  • After 1000, quality control
  • After 26000, full analysis
• If good:
  • Determine crystal structure
  • Return crystal structure
Coding the continuum: Serial crystallography

1 msec = 50 km
200 msec = 10 000 km
12 000 msec = 600 000 km
[moon = 384 000 km]
600K msec = 30 Mkm
[L1 point = 1.5M km]

1 image/20 msec
1 image/15 sec
Multiple chips @ 7 min each

6 MB, 5 msec
6 GB, 1 sec
160 GB, 60 sec
0.2-1 TB 3000 sec

6 MB, 5 msec
6 GB, 1 sec
160 GB, 60 sec
0.2-1 TB 3000 sec
Advanced Photon Source

Argonne Leadership Computing Facility

1 km
10 μsec
RTT
Similar needs arise across modern (AI-enabled) science
Learned Function Accelerators (LFAs)

Physics Code

```python
@lfa(model='ML1', executor='A21')
def func(x):
    lfa = dhub.get_model('ML1')
    if lfa.is_supported(x):
        y = lfa.run(x)
    else:
        y = func(x)
    lfa_f.update(x, y)
```

AI Engine

- **UQ Engine**
  \[(X, f) \rightarrow \sigma\]

- **Inference Engine**
  \[(X, f) \rightarrow y\]

- **Training Engine**
  \[(X, y) \rightarrow f\]

DOE Infrastructure

- **Model Library**
- **AI Accelerators**
- **Data Storage**
Coding the continuum: Closed solution

https://read.acloud.guru/aws-greengrass-the-missing-manual-2ac8df2fbd4f
Coding the continuum:
Elements of an open solution

Thanks to colleagues, especially:

Rachana Ananthakrishnan
Yadu Babuji
Ben Blaiszik
Kyle Chard
Ryan Chard
Zhuozhao Li
Tyler Skluzeck
Steve Tuecke
Anna Woodard
Logan Ward

DLHub
Parsl
SCRIMP
funcX
Data services
Auth
Coding the continuum: Elements of an open solution

```python
# App that estimates pi by placing points in a box
@example
@python_app
def pi(total):
    import random

    # Set the size of the box (edge length) in which we drop random points
    edge_length = 10000
    center = edge_length / 2
    c2 = center ** 2
    count = 0

    for i in range(total):
        # Drop a random point in the box.
        x,y = random.randint(1, edge_length),random.randint(1, edge_length)
        # Count points within the circle
        if (x-center)**2 + (y-center)**2 < c2:
            count += 1

    return (count*4/total)

# App that computes the average of the values
@example
@python_app
def avg_points(a, b, c):
    return (a + b + c)/3

# Estimate three values for pi
a, b, c = pi(10**6), pi(10**6), pi(10**6)
# Compute the average of the three estimates
avg_pi = avg_points(a, b, c)
```


http://parsl-project.org
Coding the continuum: Elements of an open solution

```python
In [1]: from funcx_sdk.client import FuncXClient
   fxc = FuncXClient()

In [2]: def add(data):
   ...:     sum_val = sum(data['data'])
   ...:     return sum_val

In [3]: fxc.register_function("add_func", func, description="Sum a list of numbers.")

In [4]: input_data = [1, 2, 3]
   res = fxc.run(input_data, "user#laptop", "add_func")

In [5]: print(res)
   6
```

**Portable code**
- **Python**: Docker, Shifter, Singularity

**Any access**
- SSH, Globus, cluster or HPC scheduler

**Any computer**
- Clusters, clouds, HPC, accelerators
**funcX**: Transform clouds, clusters, and supercomputers into high-performance function serving systems

```python
In [1]: from funcx_sdk.client import FuncXClient
    fxc = FuncXClient()

In [2]: func = ""
    def add(data):
        sum_val = sum(data['data'])
        return sum_val
    ""

In [3]: fxc.register_function("add_func", func, description="Sum a list of numbers.")

In [4]: input_data = [1, 2, 3]
    res = fxc.run(input_data, "user#laptop", "add_func")

In [5]: print(res)
6
```

Simply deploy **funcX** endpoint to transform a computer into a function serving system.
**funcX**: Transform clouds, clusters, and supercomputers into high-performance function serving systems

```python
In [1]: from funcx_sdk.client import FuncXClient
   fxc = FuncXClient()

In [2]: def add(data):
    sum_val = sum(data['data'])
    return sum_val

In [3]: fxc.register_function("add_func", func, description="Sum a list of numbers.")

In [4]: input_data = [1, 2, 3]
   res = fxc.run(input_data, "user#laptop", "add_func")

In [5]: print(res)
```

Simply deploy **funcX** endpoint to transform a computer into a function serving system.
Latency (s) for functions running on ALCF Cooley cluster, submitted from login node

Strong scaling
Weak scaling

(a) tabular file extraction  (b) MNIST digit prediction  (c) DIALS stills process  (d) tomographic preview  (e) correlation spectroscopy
## Common FaaS systems, compared

<table>
<thead>
<tr>
<th></th>
<th>Function Language</th>
<th>Intended Infrastructure</th>
<th>Virtualization</th>
<th>Triggers</th>
<th>Maximum Walltime (s)</th>
<th>Billing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Lambda</strong></td>
<td>C#, Go, Java, Powershell, Ruby Python, Node.js</td>
<td>Public cloud, Edge (Greengrass)</td>
<td>Firecracker (KVM)</td>
<td>HTTP, AWS services</td>
<td>900</td>
<td>Requests, runtime, memory</td>
</tr>
<tr>
<td><strong>Google Cloud Functions</strong></td>
<td>BASH, Go, Node.js, Python</td>
<td>Public cloud</td>
<td>Undefined</td>
<td>HTTP, Pub/Sub, storage</td>
<td>540</td>
<td>Requests, runtime, memory</td>
</tr>
<tr>
<td><strong>Azure Functions</strong></td>
<td>BASH, Java, Python, Visual Studio</td>
<td>Public cloud, local</td>
<td>OS images</td>
<td>HTTP, APIM, MS services</td>
<td>600</td>
<td>Requests, runtime, SLA</td>
</tr>
<tr>
<td><strong>OpenWhisk</strong></td>
<td>Ballerina, Go, Java, Node.js, Python, Go</td>
<td>Kubernetes, Private cloud, Public cloud</td>
<td>Docker</td>
<td>HTTP, IBM Cloud OW-CLI</td>
<td>300</td>
<td>IBM Cloud: Requests, runtime Local: NA</td>
</tr>
<tr>
<td><strong>Kubeless</strong></td>
<td>Node.js, Python .NET, Ruby Ballerina, PHP</td>
<td>Kubernetes</td>
<td>Docker</td>
<td>HTTP, scheduled, Pub/Sub</td>
<td>Undefined</td>
<td>NA</td>
</tr>
<tr>
<td><strong>SAND</strong></td>
<td>C, Go, Java, Node.js, Python</td>
<td>Public cloud, Private cloud</td>
<td>Docker</td>
<td>HTTP, Internal event</td>
<td>Undefined</td>
<td>Triggers</td>
</tr>
<tr>
<td><strong>Fn</strong></td>
<td>Go, Java, Ruby, Node.js, Python</td>
<td>Public cloud, Kubernetes</td>
<td>Docker</td>
<td>HTTP, direct trigger</td>
<td>300</td>
<td>NA</td>
</tr>
<tr>
<td><strong>Abaco</strong></td>
<td>Container</td>
<td>TACC clusters</td>
<td>Docker</td>
<td>HTTP</td>
<td>Undefined</td>
<td>Undefined</td>
</tr>
<tr>
<td><strong>funcX</strong></td>
<td>Python</td>
<td>HPC, Public cloud</td>
<td>Singularity, Shifter, Docker</td>
<td>HTTP, Globus Automate</td>
<td>No limit</td>
<td>HPC SUs</td>
</tr>
</tbody>
</table>
Coding the continuum: Elements of an open solution

Incremental construction of a **personalized cost map**

- Build black-box performance models from observed execution times for different codes on different platforms
- Transfer learning across codes, problem sizes, and hardware platforms
- Experiment design to choose experiments that maximize reduction in uncertainty
- Evolve models over time as codes and platforms change
- Use models for instance selection and scheduling
Example: A cost map for bioinformatics applications on different AWS instance types

On average, within 30% of final error after 4 experiments and within 2.3% after 6
Coding the continuum:
Elements of an open solution

Detect and respond to events
→ E.g., in HPC file systems: FSMon (Arnab Paul et al.)
Invoke RESTful services, and accept user input
Manage short- and long-lived activities
Flow automation in a neuroanatomy automation

1. Image
2. Acquire
3. Pre-process

4. Preview & center
5. User: Validate & input

6. Reconstruct
7. Publish
8. Visualize

9. Science!

Advanced Photon Source

ALCF

UChicago

Compute Lab
Coding the continuum: Elements of an open solution

Cloud-hosted services support data lifecycle events

→ Cloud for high-reliability, modest-latency actions
→ Integrated OAuth-based security with delegation
Coding the continuum: Elements of an open solution

DLHub
Data and Learning Hub for Science
A simple way to find, share, publish, and run machine learning models and discover training data for science

Get Started

1. Describe
2. Publish
3. Run

https://arxiv.org/abs/1811.11213
Paper @ Session 7, 1:30pm today
dlhub.org
Coding the Continuum: Thanks for support

US Department of Energy
US National Science Foundation
US National Institutes of Health
US National Institute of Standards and Technology
Amazon Web Services
Globus subscribers
“the machine disintegrates across the net into a set of special-purpose appliances”

Coding the [location-speed] continuum

**Code**: verb:

1) to arrange or enter in a code

“Henceforth, **location** for itself, and **speed** for itself shall completely reduce to a mere shadow, and only some sort of union of the two shall preserve independence.”

2) to write code for

Distribute computational tasks across a heterogeneous computing fabric

\[
\frac{T}{\sqrt{\pi D^2/X^2}} + 2D = B
\]

<table>
<thead>
<tr>
<th>Cost map</th>
<th>SCRIMP</th>
<th>Data fabric</th>
<th>Data services</th>
<th>Write programs</th>
<th>Parsl</th>
<th>Model registry</th>
<th>DLHub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function fabric</td>
<td>funcX</td>
<td>Trust fabric</td>
<td>Auth</td>
<td></td>
<td></td>
<td>Flows</td>
<td>Automate</td>
</tr>
</tbody>
</table>

foster@anl.gov

labs.globus.org – dlhub.org – globus.org – parsl-project.org