General-purpose Parallel and Distributed Programming Systems at Scale

Dr. Tsung-Wei Huang

Department of Electrical and Computer Engineering

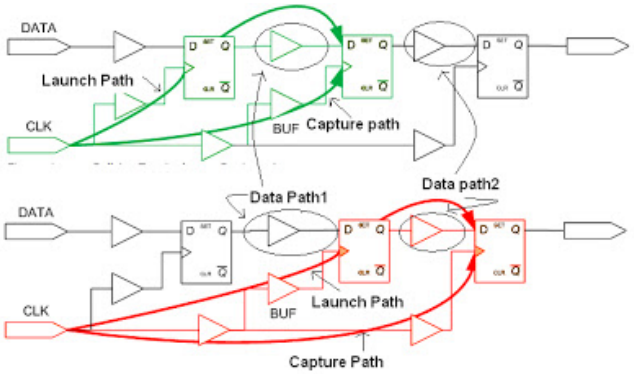
University of Utah, Salt Lake City, UT

Outline

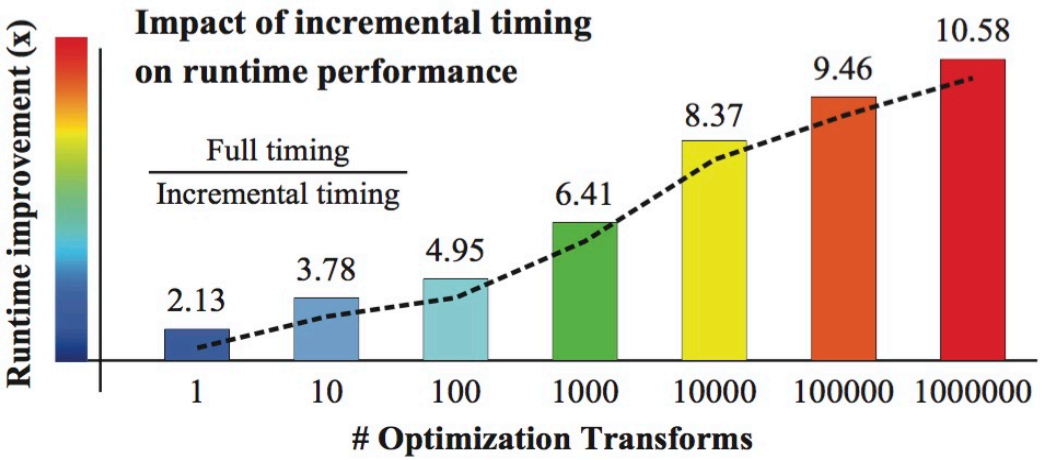
- **My early research at UIUC**
  - OpenTimer: A VLSI timing analysis tool
  - Express parallelism in the right way
- **A general-purpose distributed programming system**
  - DtCraft and its ecosystem
  - Machine learning, graph algorithms, chip design
  - Cpp-Taskflow: Modern C++ parallel task programming
- **Conclusion and future work**

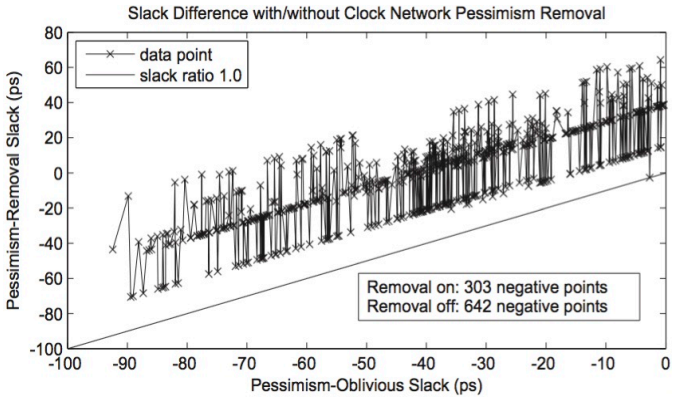
My Early Research at UIUC

- **Static timing analysis (STA)**
  - Key component in VLSI design
  - Verify the circuit timing
- **Many challenges**
  - Incremental timing
  - Parallelization, scalability



Setup/hold timing checks

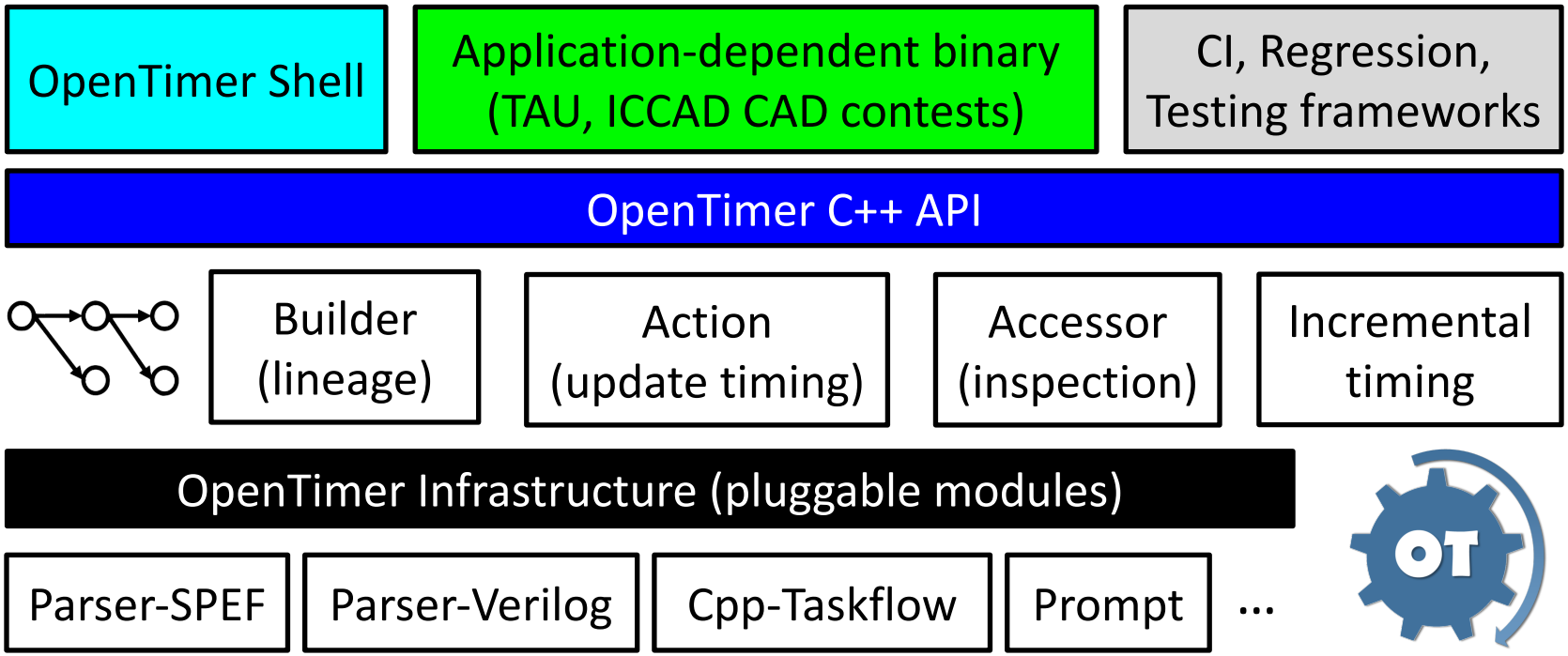




Path-based timing analysis to reduce pessimism

OpenTimer: An Open-source STA Tool

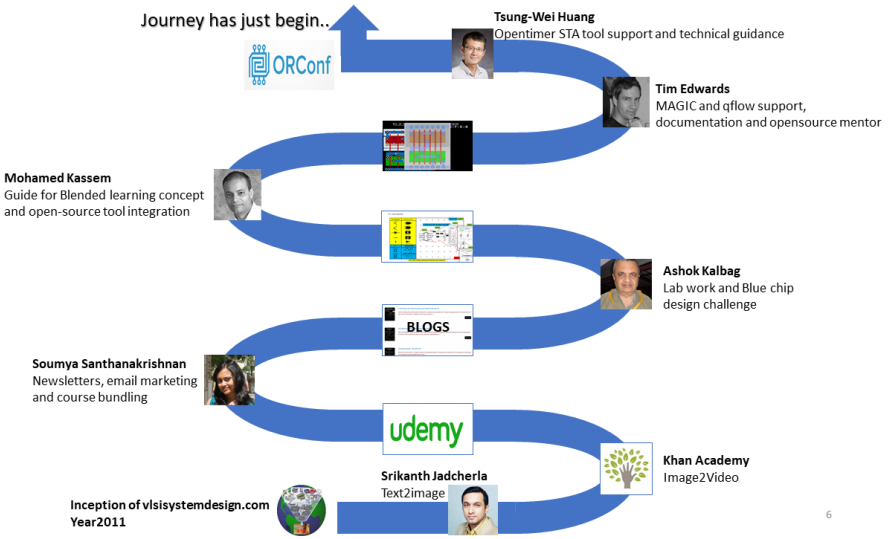
- TAU14 (1st place), TAU15 (2nd place), TAU16 (1st place)
- Best tool awards (WOSET ICCAD18, LibreCore16)
- Golden timers in VLSI and CAD communities



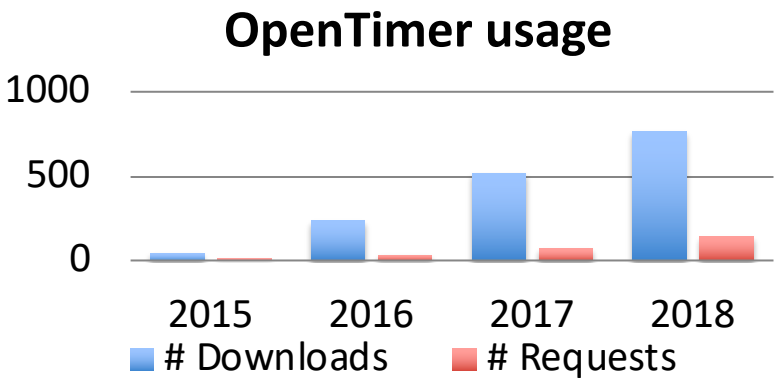
T.-W. Huang et al., "OpenTimer: A High-performance Timing Analysis Tool," IEEE/ACM ICCAD15

OpenTimer Community – Thank You!

OpenTimer is begin used in many projects



Collaboration with VSD (start-up) and Udemy online education platform



We are growing! ***> 500 downloads in 2018***

Song Chen *Contest Committee Member*

Xin Zhao *Contest Committee Member*

Xi Chen *Contest Committee Member*

Debjit Sinha *Workshop General Chair*

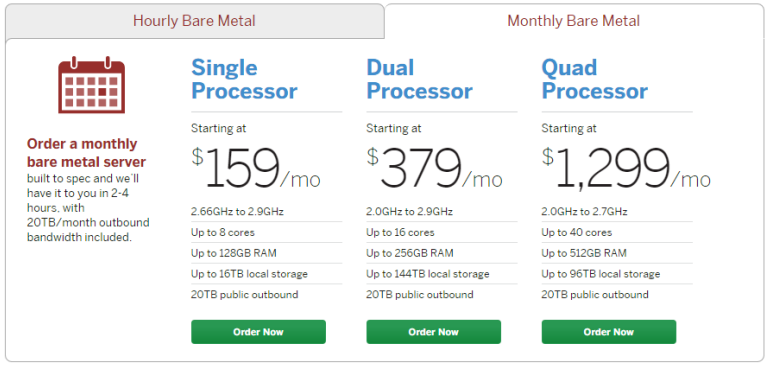
Qiuyang Wu *Workshop Technical Chair*

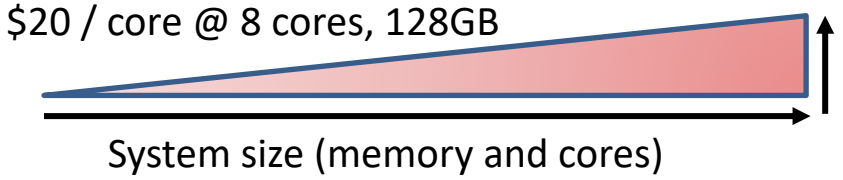
Tsung-Wei Huang *OpenTimer Support*

Acknowledged by ACM TAU16-19 Contests

Collaboration with IBM

- ***"We need a new distributed timing analysis method to deal with the large design complexity,"* IBM Timing Group, Fishkill, NY, 2015**
  - Existing tools are architecturally limited by one machine
  - Can take *800GB RAM* and *14 days* on a single machine
  - Maintaining "bare-metal" machines is not cost-efficient





Objective: Execute the same # computations, distributed across an array of less expensive per unit cost cloud systems

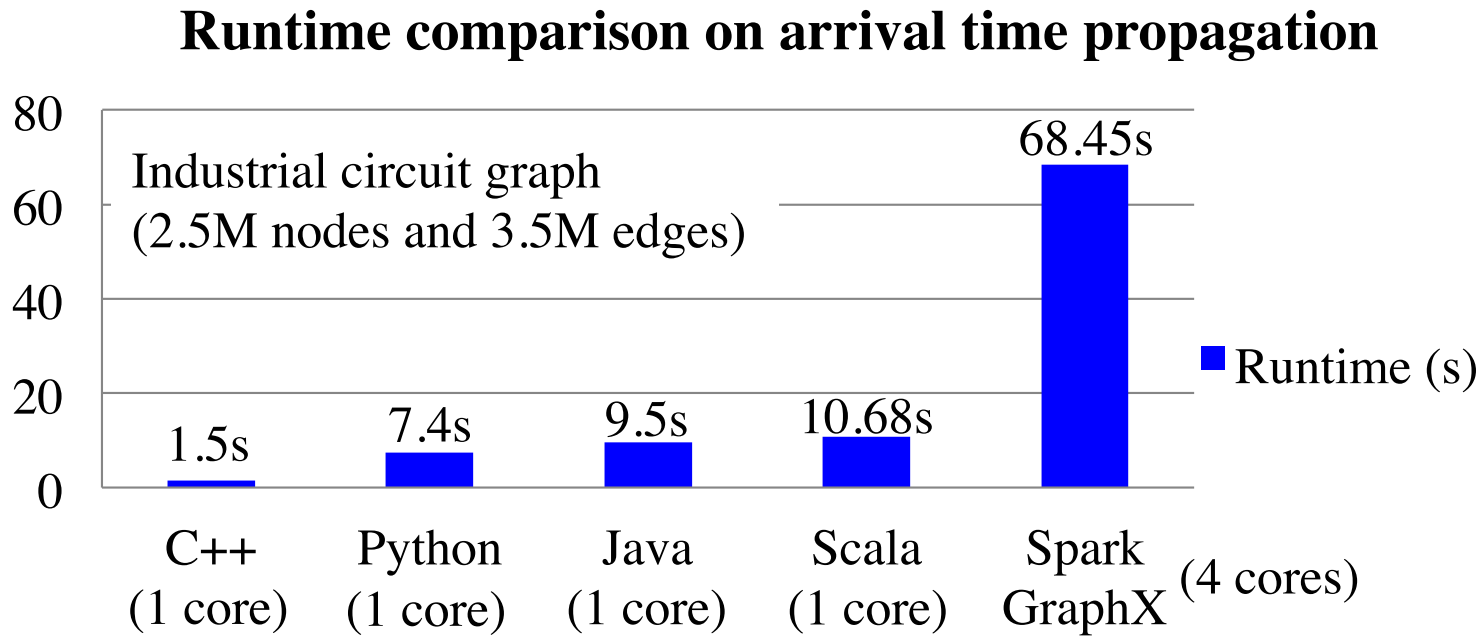
Leverage economy of scale by decreasing the per unit cost of computation "hill"

Existing Tools in Cloud Computing

- **Hadoop**
  - Distributed MapReduce platform on HDFS
- **Zookeeper**
  - Coordination service for distributed application
- **Mesos**
  - A high-performance cluster manager
- **Spark**
  - A general computing engine for big-data analytics
- **…**

Are these tools suitable for our application?

Big-data Tools is NOT an Easy Fit …

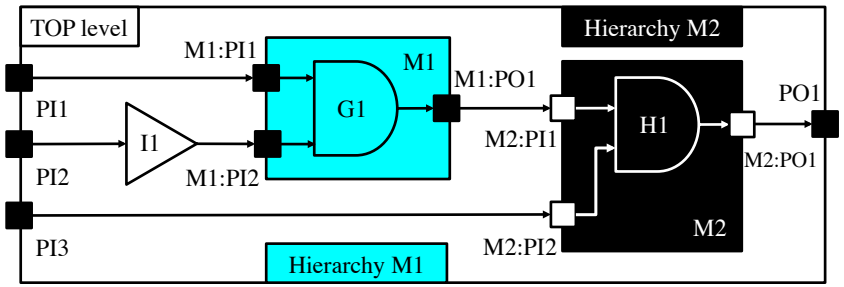


Method	Spark 2.0 (RDD + GraphX Pregel)	Java (SP)	C++ (SP)
Runtime (s)	68.45	9.5	**1.50**

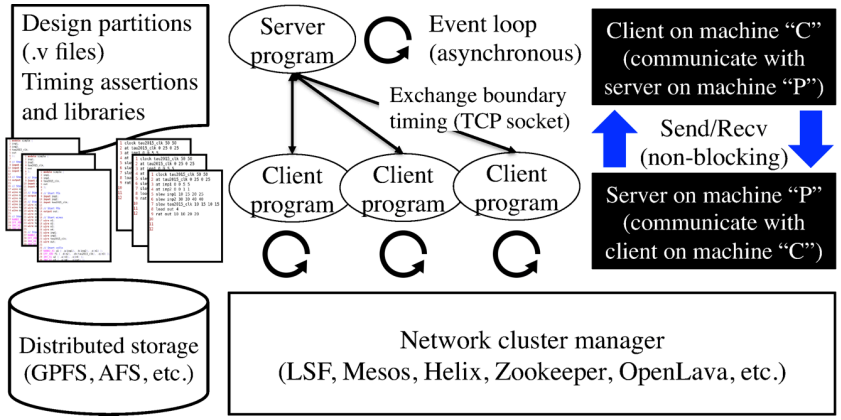
MapReduce overhead *Language overhead*

If it is Hard, let me Hard-code it …

- **A "specialized" distributed timer**
  - Low-level Linux sockets and message passing code
  - Explicit resource partition and mapping
- **Single-server multiple-client model**
  - Server coordinates with clients to exchange data



Three partitions, top-level, M1, and M2 (given by design teams)



T.-W. Huang, et al, "A Distributed Timing Analysis Framework for Large Designs," IEEE/ACM DAC16
T.-W. Huang, et al, US Patent US20170242945A1, assignee IBM, 2017
T.-W. Huang, et al, US Patent US9836572B2, assignee IBM, 2017

Observations

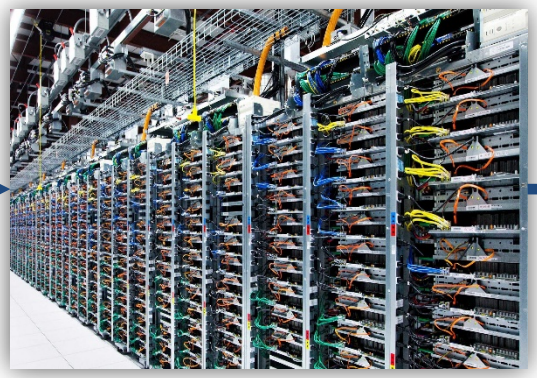
- Existing big-data tools have large room to improve
- Fast prototype but hard long-term maintenance
- Non-unified tooling environment
- Cumbersome software dependencies
- Cannot afford hard-coded solution all the time
- Difficult low-level concurrency controls
- Hand-written message passing code
- …

We want a general-purpose programming system to streamline the building of parallel/distributed systems!

DARPA's Electronic Resurgence Initiative

- **$1.5B investment to reduce the design complexity**
  - People without chip design experience can design chips

~~Chip layout army~~



Massive cloud computing!

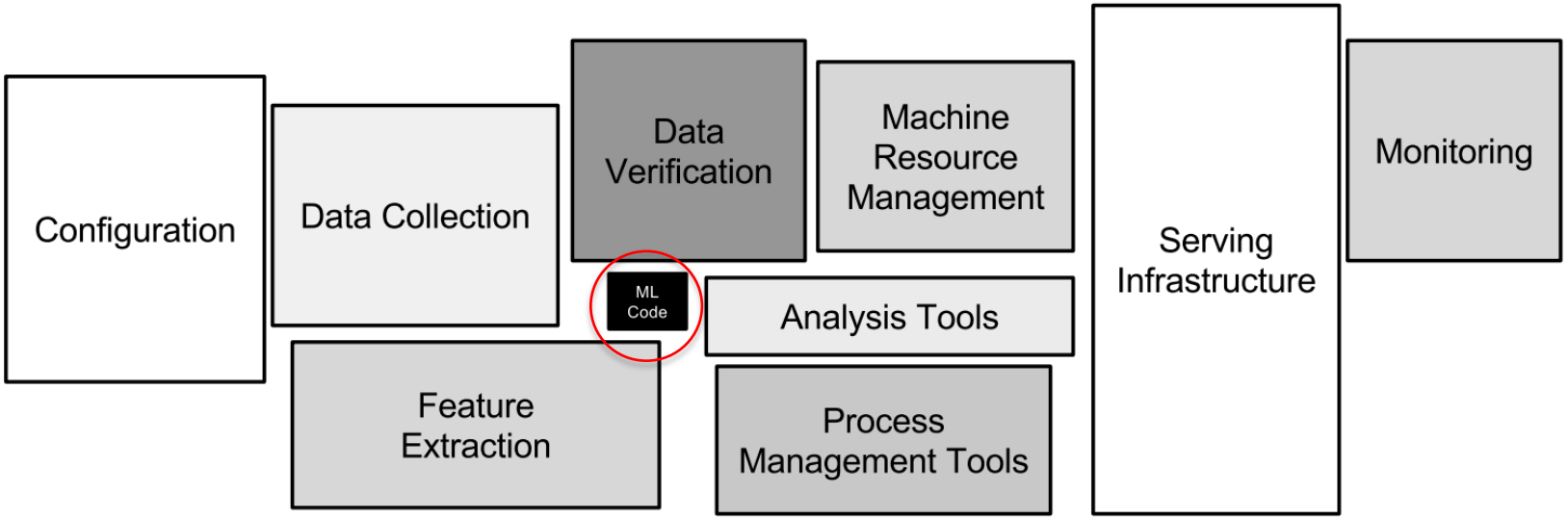
No-touch foundry

- **A general-purpose silicon compiler (IDEA & POSH)**
  - fully-automated layout generator, no human in the loop
  - Massive software efforts and tool integrations
  - *Parallel/distributed computing and machine learning**

**DARPA IDEA/POSH Kick-off Meeting, Boston, June, 2018*

Hidden Technical Debts in ML Systems

- **Only a TINY fractions of ML systems is ML code**
  - Writing ML code is cheap; system maintenance is $$$$$
- **ML innovations are moving to "system innovations"**
  - Spark MLflow, TensorFlow 2.0, PyTorch, MindsDB
  - *Parallel/distributed computing, transparency, scalability*



Google Inc, "Hidden Technical Debt in Machine Learning Systems," NIPS 2017

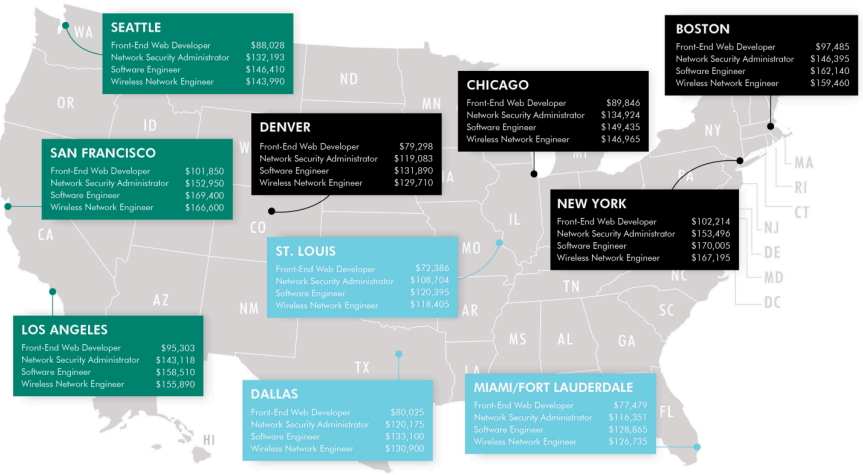
Keep Programmability in Mind

- **In the cloud era …**
  - Hardware is just a commodity
  - Building a cluster is cheap
  - Coding takes people and time

Google Cloud

TECH SALARIES ACROSS THE U.S.

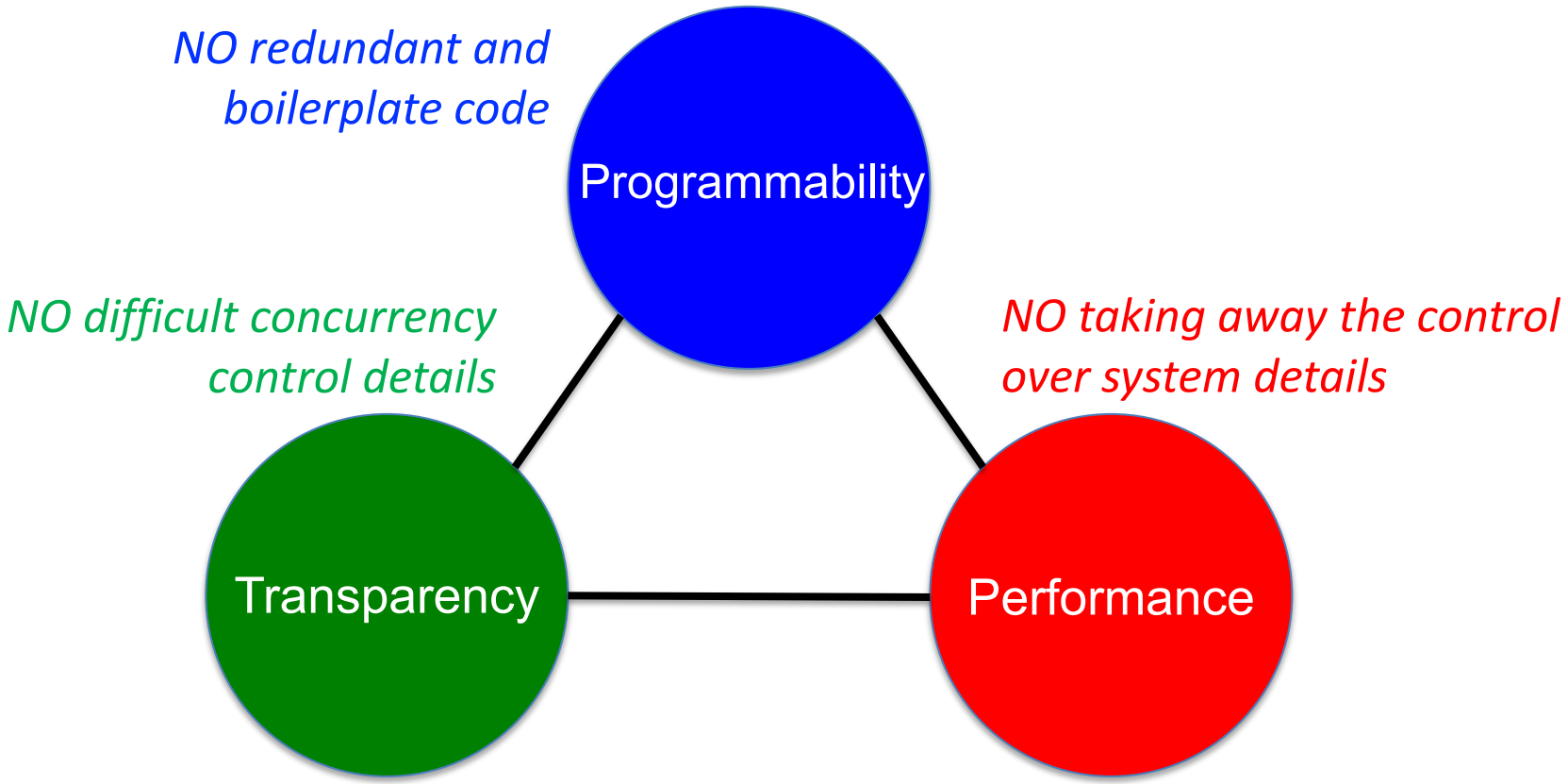
Compare midpoint starting salaries for technology positions in different cities.*



Programmability can affect the "performance" and "productivity" in all aspects (details, styles, high-level decisions, etc)!

2018 Avg Software Engineer salary (NY) > $170K

Want a Unified Programming System

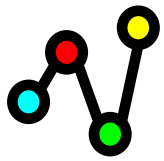


"We want to let users easily express their parallel/distributed computing workload without taking away the control over system details to achieve high performance, using our expressive API written in modern C++"

Outline

- **My early research at UIUC**
  - OpenTimer: A VLSI timing analysis tool
  - Express parallelism in the right way
- **A general-purpose distributed programming system**
  - DtCraft and its ecosystem
  - Machine learning, graph algorithms, chip design
  - Cpp-Taskflow: Modern C++ parallel task programming
- **Conclusion and future work**

A New Solution: DtCraft



Application pipeline (domain-specific applications)

MLCraft (distributed machine learning library)

GraphCraft (distributed graph processing library)

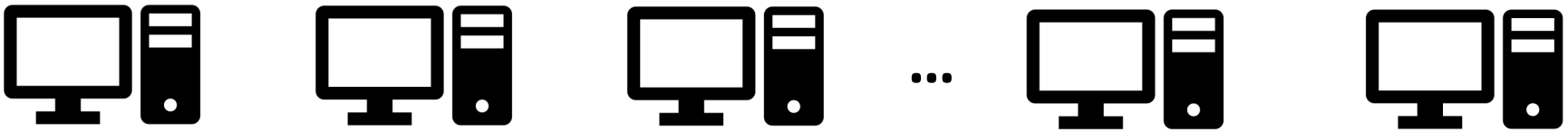
...

DataCraft (distributed data processing server)

DtCraft C++ Programming Runtime

Cpp-Taskflow (Parallel task programming library)

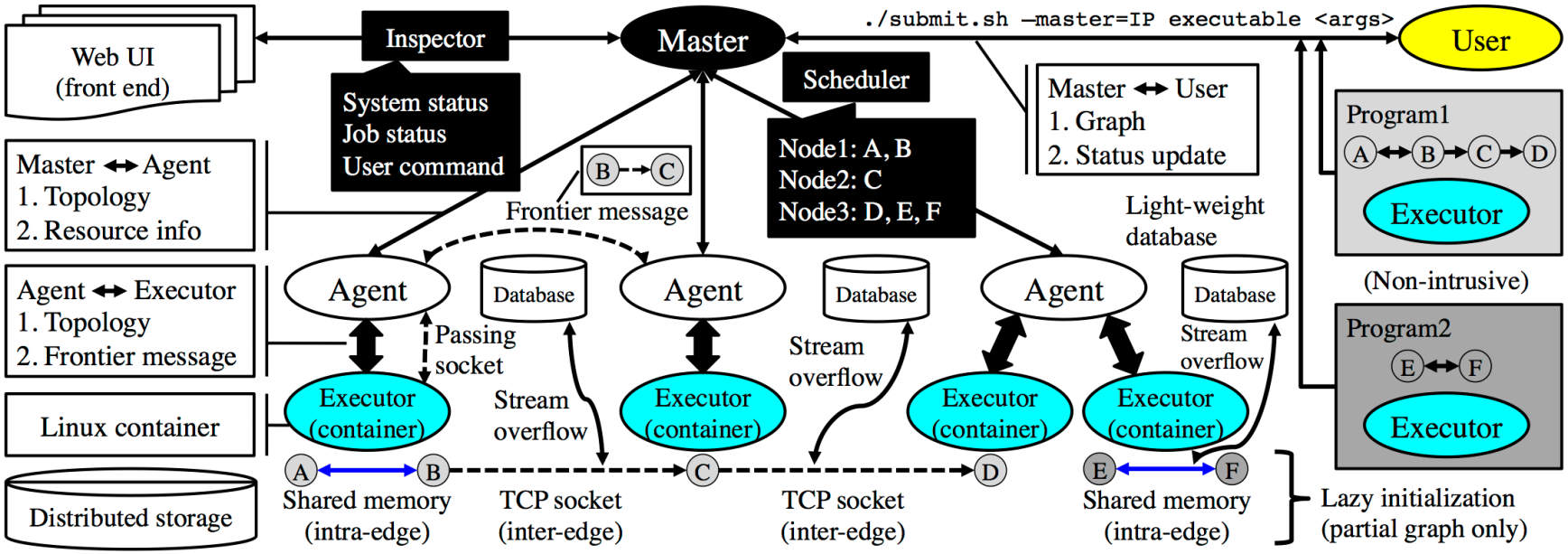
Cpp-Container (Docker, RunC, LibContainer, LXC)



DARPA IDEA Grant, NSF Grant, Best Open-source Tool Awards (IEEE IPDPS, ACM MM, WOSET, CPPConf, ACM DEBS, ACM/IEEE DAC, IEEE/ACM ICCAD, IEEE TCAD, ACM TAU, US Patents, etc.)

DtCraft C++ Programming Runtime

- **You describe an application in a *stream graph***
  - "No arm wrestling" with difficult concurrency details
- **Everything is by default distributed**
  - Same program runs on one and many machines



A Hello-World Example

- **An iterative and incremental flow**
  - Two vertices + two streams

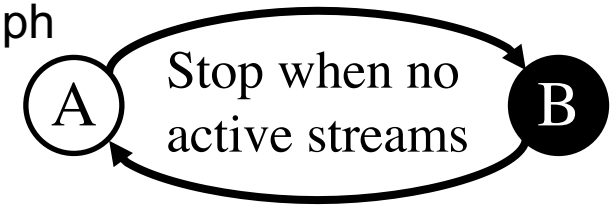
Step 4: A's resource
1 CPU / 1 GB RAM

Step 2:
string msg;

Step 3: A➔B callback

```
[=] (auto& B, auto& is) {
  Extract string from is;
  print string;
}
```

Step 1: stream graph



istream ➔ B

Step 3: A➔B callback

```
[=] (auto& A, auto& is) {
  Extract string from is;
  print string
}
```

Step 2:
string msg;

Step 4: B's resource
1 CPU / 1 GB RAM

A ⬅ istream

Step 5: ./submit –master=127.0.0.1 hello-world

DtCraft Code of Hello World

```
dtc::Graph G;

auto A = G.vertex();
auto B = G.vertex();

G.container().add(A).cpu(1).memory(1_GB);

auto AB = G.stream(A, B).on(
  [] (dtc::Vertex& B, dtc::InputStream& is) {
    if(std::string b; is(b) != -1) {
      dtc::cout("Received: ", b, '\n');
      return dtc::Event::REMOVE;
    }
    return dtc::Event::DEFAULT;
  }
);

auto BA = G.stream(B, A);

A.on([&AB] (dtc::Vertex& v) {
  (*v.ostream(AB))("hello world from A"s);
  dtc::cout("Sent 'hello world from A' to stream ", AB, "\n");
});

G.container().add(B).cpu(1).memory(1_GB);

B.on([&BA] (dtc::Vertex& v) {
  (*v.ostream(BA))("hello world from B"s);
  dtc::cout("Sent 'hello world from B' to stream ", BA, "\n");
});

BA.on([] (dtc::Vertex& A, dtc::InputStream& is) {
  if(std::string a; is(a) != -1) {
    dtc::cout("Received: ", a, "\n");
    return dtc::Event::REMOVE;
  }
  return dtc::Event::DEFAULT;
});

dtc::Executor(G).run();
```

- **Only a couple lines of code**
- Single sequential program
- No explicit data management
- Distributed across computers
- Easy-to-use interface
- Asynchronous by default
- Scalable to many threads
- Scalable to many machines
- In-context resource controls
- Transparent concurrency
- Automatic Linux container

… and more

Without DtCraft …

```cpp
auto count_A = 0;
auto count_B = 0;

// Send a random binary data to fd and add the
// received data to the counter.
auto pinpong(int fd, int& count) {
  auto data = random<bool>()
  auto w = write(fd, &data, sizeof(data));
  if(w == -1 && errno != EAGAIN) {
    throw system_error("Failed on write");
  }
  data = 0;
  auto r = read(fds, &data, sizeof(data));
  if(r == -1 && errno != EAGAIN) {
    throw system_error("Failed on read");
  }
  count += data;
}

int fd = -1;
std::error_code errc;
```

server.cpp

```cpp
if(getenv("MODE") == "SERVER") {
  fd = make_socket_server_fd("9999", errc);
}
else {
  fd = make_socket_client_fd("127.0.0.1", "9999", errc);
}
```

client.cpp

```cpp
if(fd == -1) {
  throw system_error("Failed to make socket");
}
```

Branch your code to server and client for distributed computation!

simple.cpp → server.cpp + client.cpp

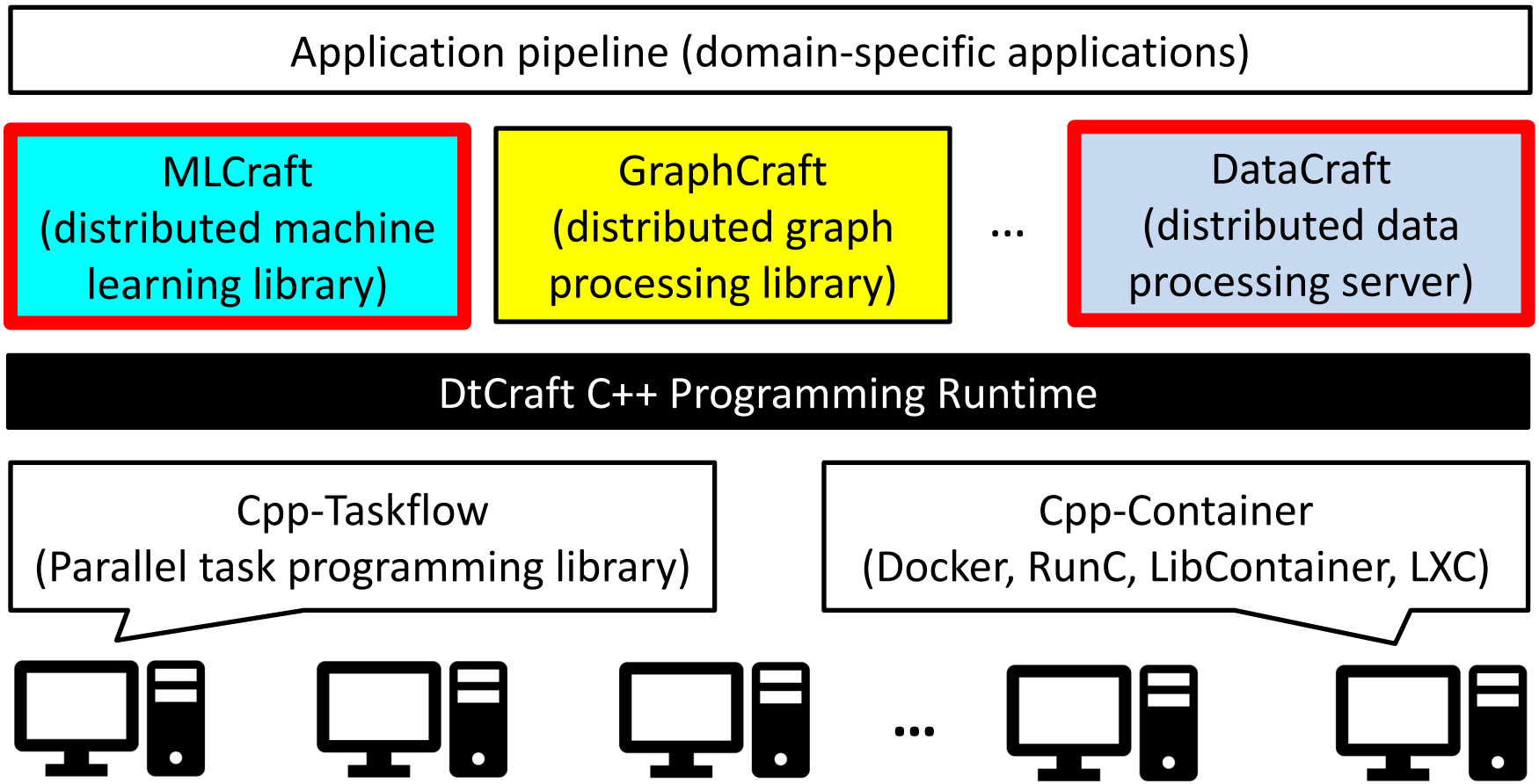
```cpp
int make_socket_server_fd(
  std::string_view port,
  std::error_code errc
) {
  int fd {-1

  struct adc
  struct adc
  struct adc

  std::memse
  hints.ai_f
  hints.ai_s
  hints.ai_p
  hints.ai_f

  int one {1
  int ret;

  if((ret =
    errc = m
    return -
  }

  // Try to
  for(ptr =

    // Ignor
    if(ptr->
      goto t
    }

    if((fd =
      errc =
      goto t
    }

    ::setsoc

    if(::bir
      errc =
      goto t
    }

    if(::lis
      errc =
      goto t
    }
    else {
      break;
    }

  try_next
```

```cpp
    if(fd != -1) {
      ::close(fd);
      fd = -1;
    }
  }

  ::freeac

  // Assig
  return f
}

int make_s
  std::str
  std::str
  std::err
) noexcept

  errc.cle

  struct a
  struct a

  std::mem
  hints.ai
  hints.ai
  hints.ai

  int ret;
  int fd =
  int tri

  if((ret
    errc =
    return
  }

  // Try e
  for(auto

    // Ign
    if(pt
      goto
    }

    if((fd
      err
      goto
    }

  make_f
```

```cpp
    make_fd_close_on_exec(fd);

    tries = 3;

    issue_connect:
    ret = ::connect(fd, ptr->ai_addr, ptr->ai_addrlen);

    if(ret == -1) {
      if(errno == EINTR) {
        goto issue_connect;
      }
      else if(errno == EAGAIN && tries--) {
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
        goto issue_connect;
      }
      else if(errno != EINPROGRESS) {
        goto try_next;
      }
      errc = make_posix_error_code(errno);
    }

    // Poll the socket. Note that writable return doesn't mean it is conne
    if(select_on_write(fd, 5, errc) && !errc) {
      int optval = -1;
      socklen_t optlen = sizeof(optval);
      if(::getsockopt(fd, SOL_SOCKET, SO_ERROR, &optval, &optlen) < 0) {
        errc = make_posix_error_code(errno);
        goto try_next;
      }
      if(optval != 0) {
        errc = make_posix_error_code(optval);
        goto try_next;
      }
      break;
    }

    try_next:

    if(fd != -1) {
      ::close(fd);
      fd = -1;
    }
  }

  ::freeaddrinfo(res);

  return fd;
}
```

A lot of boilerplate code plus hundred lines of scripts to enable distributed flow…

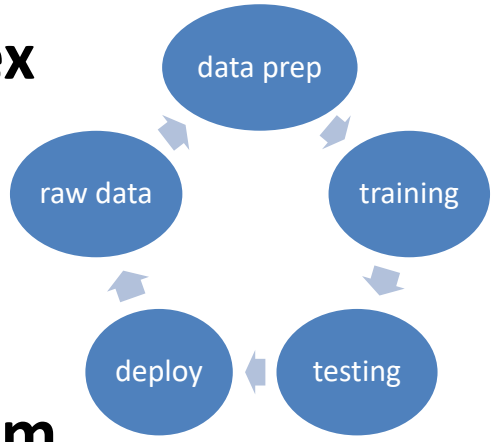
DtCraft to Handle Machine Learning

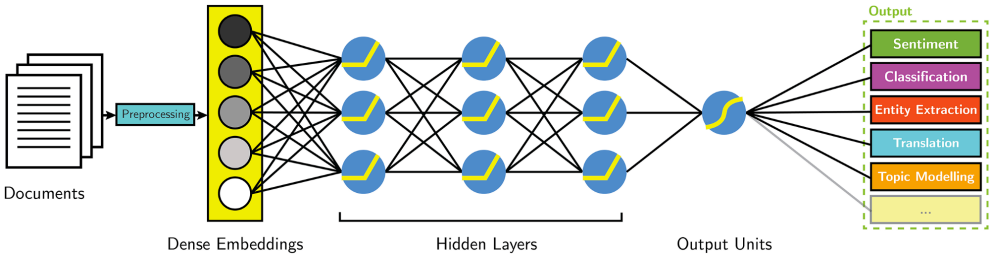


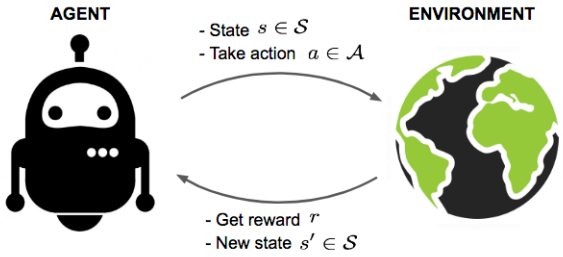
DARPA IDEA Grant, NSF Grant, Best Open-source Tool Awards (IEEE IPDPS, ACM MM, WOSET, CPPConf, ACM DEBS, ACM/IEEE DAC, IEEE/ACM ICCAD, IEEE TCAD, ACM TAU, US Patents, etc.)

MLCraft – Machine Learning at Scale

- **ML system development is very complex**
  - Hard to track experiments
  - Hard to reproduce results
  - Hard to deploy and maintain
- **MLCraft aims to be a unified ML platform**
  - Automate ML workflows and developments
  - Open API and interface to work with existing libraries
  - More robust, predictable, easier to maintain



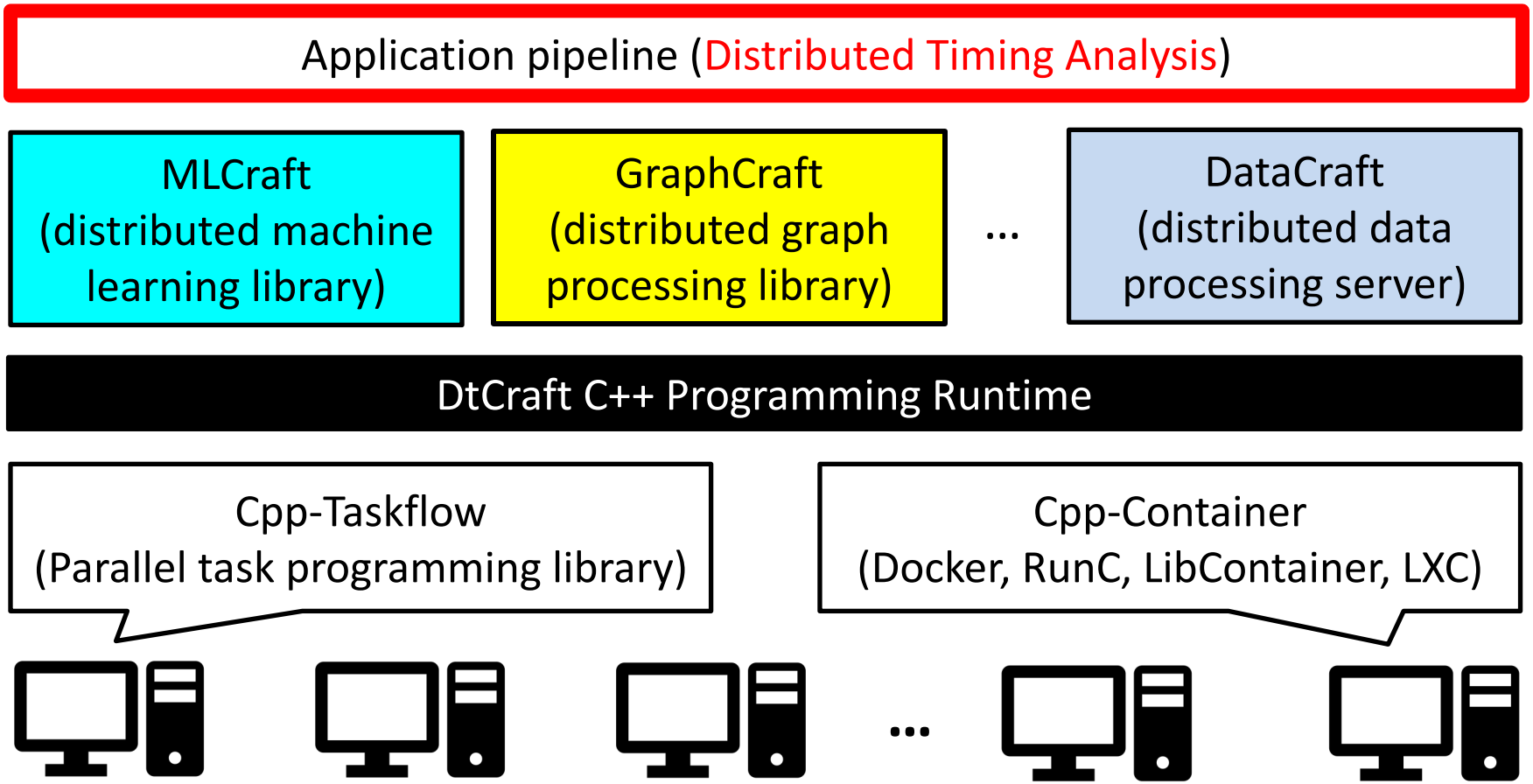




```
dtc::Graph G;

auto src = G.insert<dtc::cell::MnistStreamFeeder>(
  mnist_image_file,
  mnist_label_file,
  [] (Eigen::MatrixXf& images, Eigen::VectorXi& labels) {
    images /= 255.0f;
    return std::make_tuple(images, labels);
  }
);

src.window(1ms).frequency(1000);

auto dnn = G.insert<dtc::cell::Visitor1x1>(
  [] (dtc::ml::DnnClassifier& c) {
    dtc::cout("Creating a dnn classifier [784x30x10]\n").flush();
    c.fully_connected_layer(784, 30, dtc::ml::Activation::RELU)
     .fully_connected_layer(30, 10, dtc::ml::Activation::NONE)
     .optimizer<dtc::ml::AdamOptimizer>();
  },
  [i=0] (dtc::ml::DnnClassifier& c, std::tuple<Eigen::MatrixXf, Eigen::VectorXi>& data) mutab
    auto& [images, labels] = data;
    dtc::cout(
      "Accuracy at training cycle ", i++, " [", images.rows(), " images]: ",
      ((labels-c.infer(images)).array() == 0).count() / static_cast<float>(images.rows()), '\
    ).flush();
    c.train(images, labels, 10, 64, 0.01f, [](){});
  }
);

dnn.in(src.out());

G.container().add(dnn);
G.container().add(src);

dtc::Executor(G).run();
```

Only **“60-line”** code to automate a distributed ML workflow!

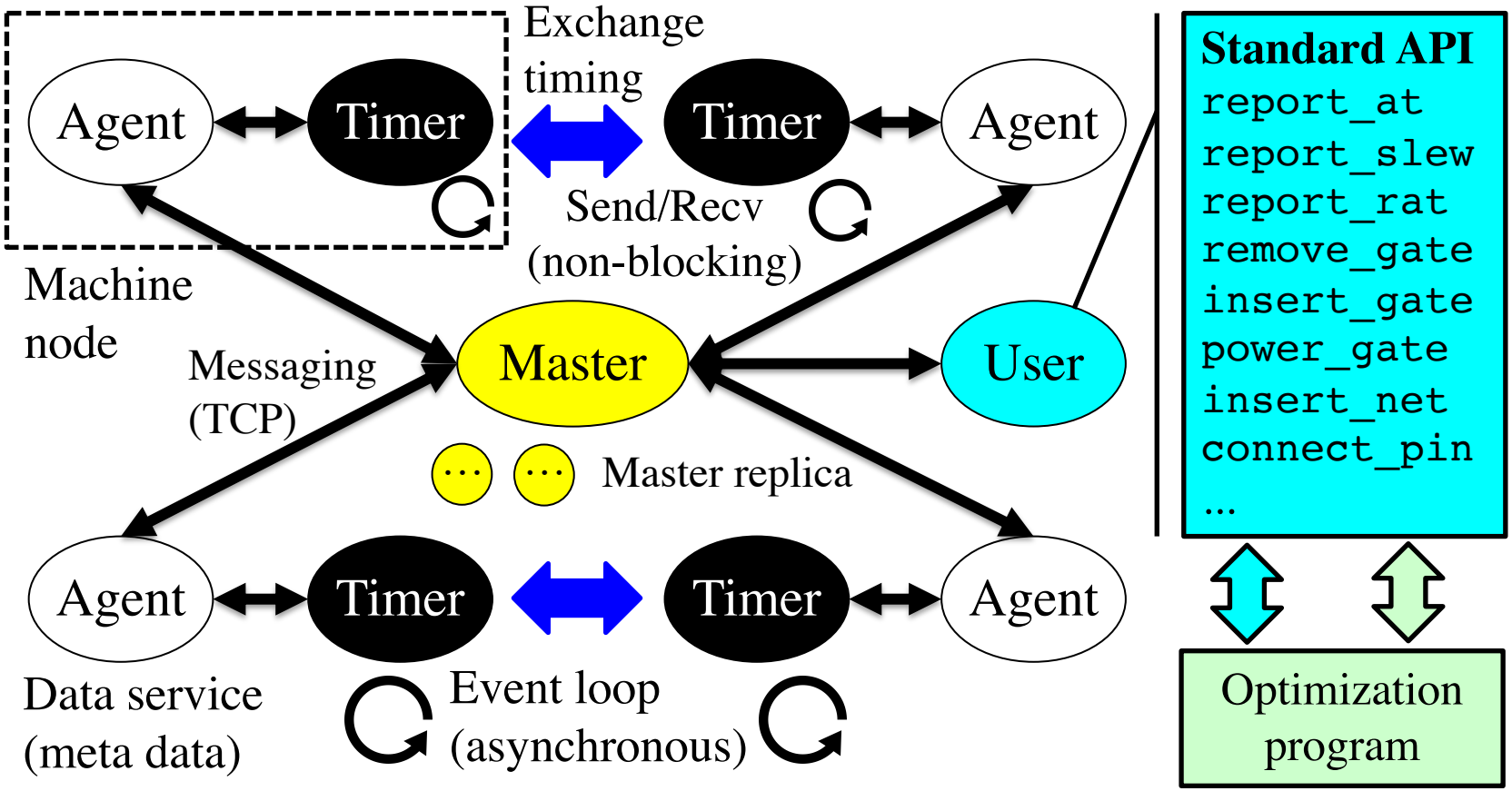
DtCraft to Handle Distributed Timing



DARPA IDEA Grant, NSF Grant, Best Open-source Tool Awards (IEEE IPDPS, ACM MM, WOSET, CPPConf, ACM DEBS, ACM/IEEE DAC, IEEE/ACM ICCAD, IEEE TCAD, ACM TAU, US Patents, etc.)

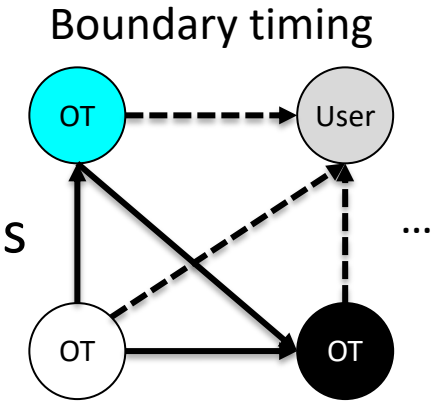
Distributed Timing Analysis

- Master coordinator and worker (a timer per partition)

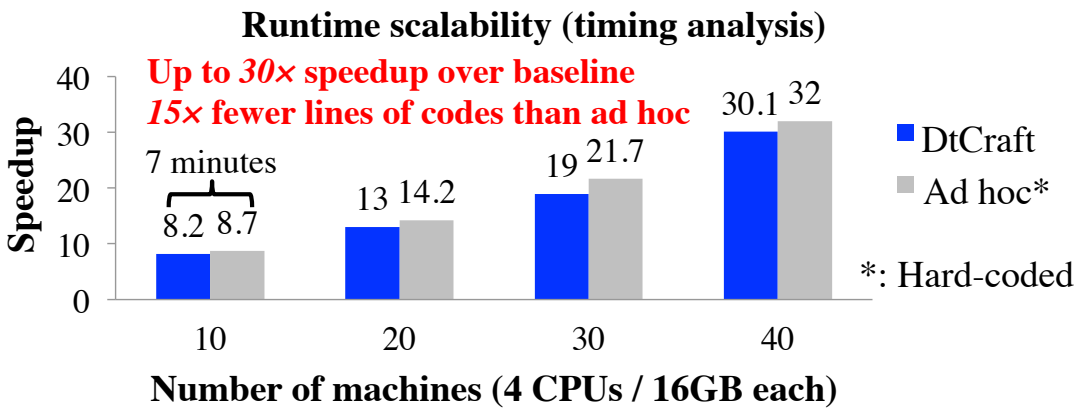


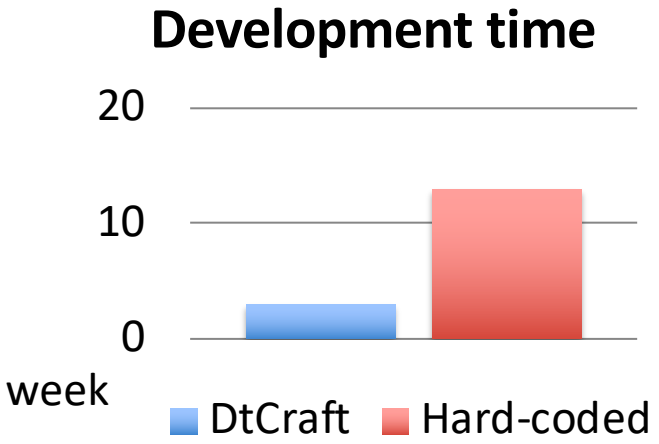
Distributed Timing Performance

- **IBM design of 250 partitions**
  - OpenTimer as the timing analyzer
  - On a cluster of 40 Amazon EC2 instances
- **Compared with hard-coded solution**
  - 15x fewer lines of code
  - Only 7% performance loss



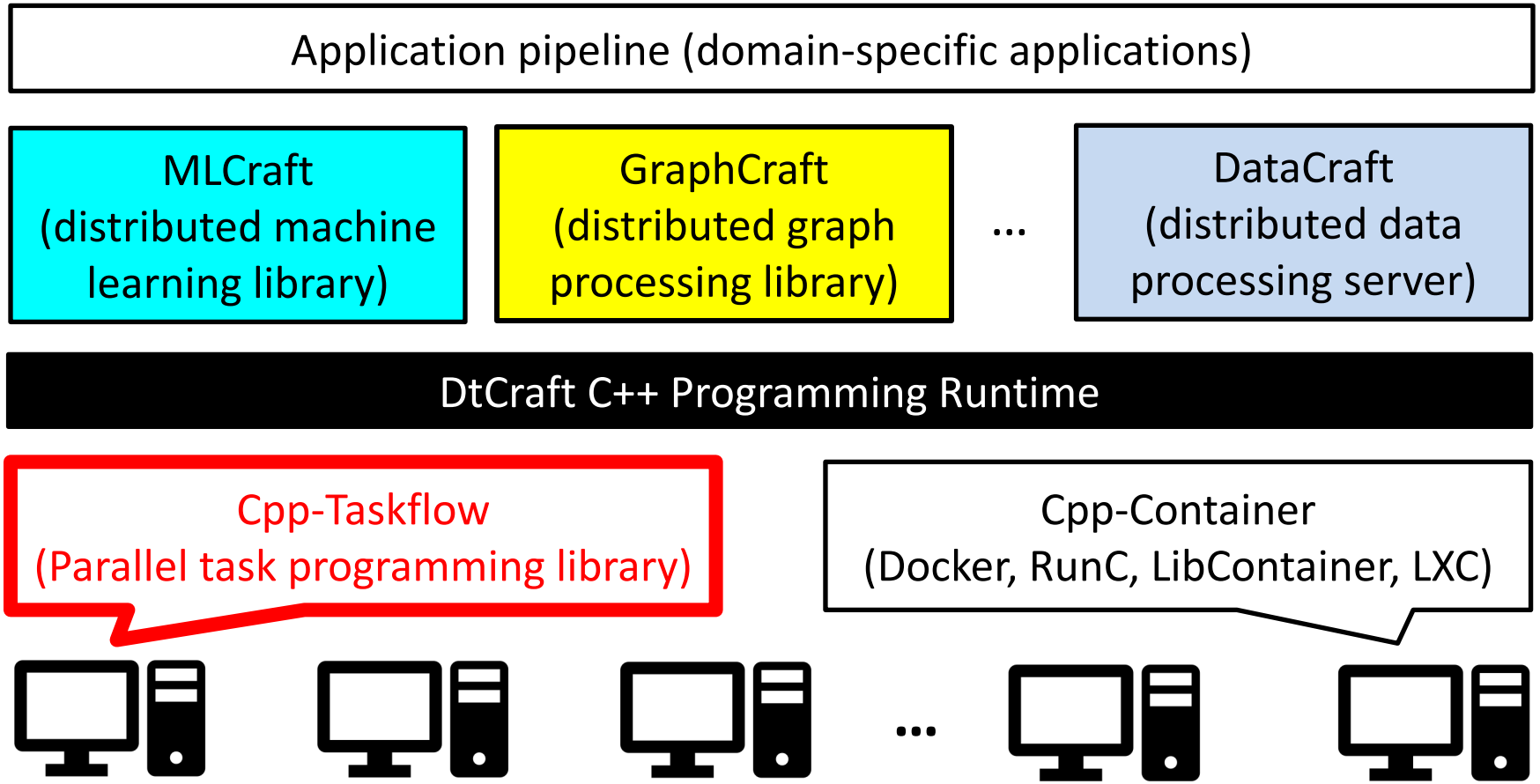
Stream graph
(250 IBM design partitions)





Improving the programmability of a system is not only making programming easier and fast, but is enabling a new innovation to solve problems more efficiently

DtCraft to Handle Low-level Concurrency



DARPA IDEA Grant, NSF Grant, Best Open-source Tool Awards (IEEE IPDPS, ACM MM, WOSET, CPPConf, ACM DEBS, ACM/IEEE DAC, IEEE/ACM ICCAD, IEEE TCAD, ACM TAU, US Patents, etc.)

Cpp-Taskflow's Project Mantra

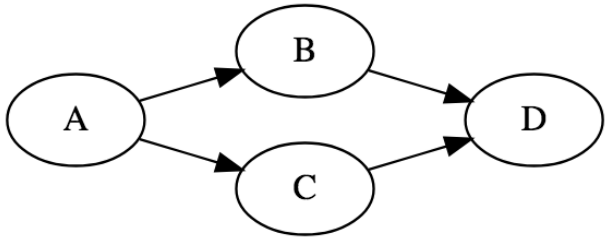
*A programming library helps developers **quickly** write **efficient** parallel programs on a **manycore machine** using **task-based** approaches in **modern C++***

- **Task-based approach scales best with multicore arch**
  - We should write tasks instead of threads
  - Not trivial due to dependencies (race, lock, bugs, etc)
- **We want developers to write parallel code that is:**
  - Simple, expressive, and transparent
- **We don't want developers to manage:**
  - Explicit thread management
  - Difficult concurrency controls and daunting class objects

Hello-World in Cpp-Taskflow

```
#include <taskflow/taskflow.hpp>   // Cpp-Taskflow is header-only
int main(){
   tf::Taskflow tf;
   auto [A, B, C, D] = tf.emplace(
      [] () { std::cout << "TaskA\n"; }
      [] () { std::cout << "TaskB\n"; },
      [] () { std::cout << "TaskC\n"; },
      [] () { std::cout << "TaskD\n"; }
   );
   A.precede(B);            // A runs before B
   A.precede(C);            // A runs before C
   B.precede(D);            // B runs before D
   C.precede(D);            // C runs before D
   tf::Executor().run(tf);  // create an executor to run the taskflow
   return 0;
}
```

*Only **15 lines** of code to get a parallel task execution!*



Hello-World in OpenMP

```
#include <omp.h>  // OpenMP is a lang ext to describe parallelism in compiler directives
int main(){
   #omp parallel num_threads(std::thread::hardware_concurrency())
   {
      int A_B, A_C, B_D, C_D;
      #pragma omp task depend(out: A_B, A_C)
      {
         s t d : : c o u t << ”TaskA\n” ;
      }
      #pragma omp task depend(in: A_B; out: B_D)
      {
         s t d : : c o u t << ” TaskB\n” ;
      }
      #pragma omp task depend(in: A_C; out: C_D)
      {
         s t d : : c o u t << ” TaskC\n” ;
      }
      #pragma omp task depend(in: B_D, C_D)
      {
         s t d : : c o u t << ”TaskD\n” ;
      }
   }
   return 0;
}
```

Task dependency clauses

Task dependency clauses

Task dependency clauses

Task dependency clauses

OpenMP task clauses are static and explicit; Programmers are responsible a proper order of writing tasks consistent with sequential execution

Hello-World in Intel's TBB Library

```
#include <tbb.h>  // Intel's TBB is a general-purpose parallel programming library in C++
int main(){
    using namespace tbb;
    using namespace tbb:flow;
    int n = task_scheduler init::default_num_threads () ;
    task scheduler_init init(n);
    graph g;
    continue_node<continue_msg> A(g, [] (const continue msg &) {
        s t d : : c o u t << "TaskA" ;
    }) ;
    continue_node<continue_msg> B(g, [] (const continue msg &) {
        s t d : : c o u t << "TaskB" ;
    }) ;
    continue_node<continue_msg> C(g, [] (const continue msg &) {
        s t d : : c o u t << "TaskC" ;
    }) ;
    continue_node<continue_msg> C(g, [] (const continue msg &) {
        s t d : : c o u t << "TaskD" ;
    }) ;
    make_edge(A, B);
    make_edge(A, C);
    make_edge(B, D);
    make_edge(C, D);
    A.try_put(continue_msg());
    g.wait_for_all();
}
```

Use TBB's FlowGraph for task parallelism

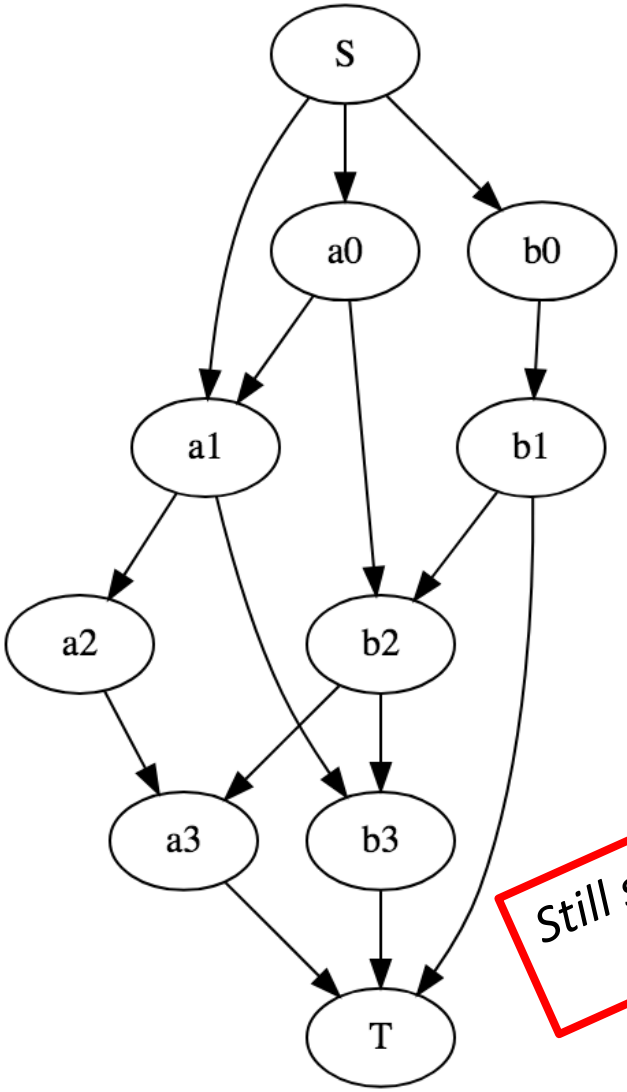
Declare a task as a continue_node

TBB has excellent performance in generic parallel computing. Its drawback is mostly in the ease-of-use standpoint (simplicity, expressivity, and programmability).

Somehow, this looks more like "hello universe" …

A Slightly More Complicated Example

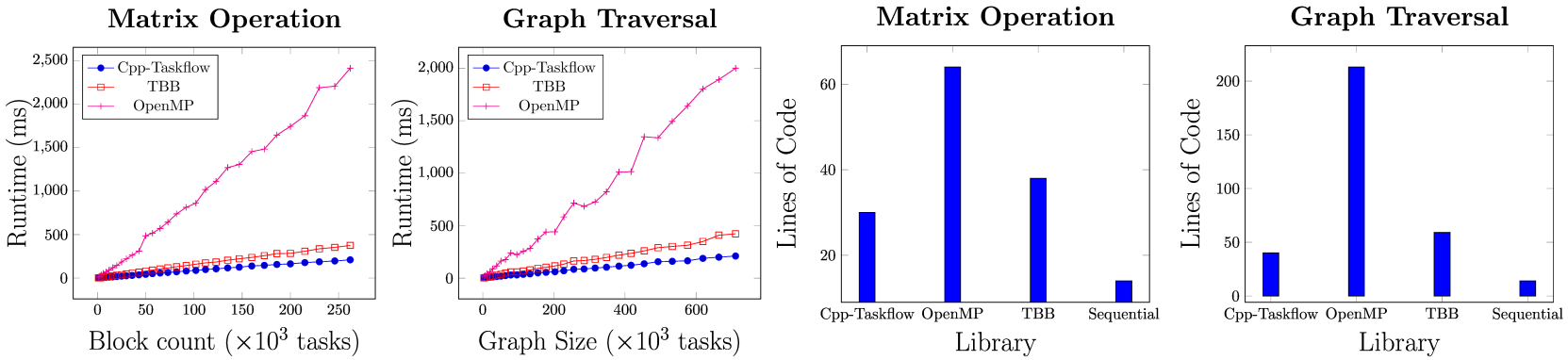
```
// source dependencies
S.precede(a0);      // S runs before a0
S.precede(b0);      // S runs before b0
S.precede(a1);      // S runs before a1
// a_ -> others
a0.precede(a1);     // a0 runs before a1
a0.precede(b2);     // a0 runs before b2
a1.precede(a2);     // a1 runs before a2
a1.precede(b3);     // a1 runs before b3
a2.precede(a3);     // a2 runs before a3
// b_ -> others
b0.precede(b1);     // b0 runs before b1
b1.precede(b2);     // b1 runs before b2
b2.precede(b3);     // b2 runs before b3
b2.precede(a3);     // b2 runs before a3
// target dependencies
a3.precede(T);      // a3 runs before T
b1.precede(T);      // b1 runs before T
b3.precede(T);      // b3 runs before T
```



Still simple in Cpp-Taskflow

Micro-benchmark Performance

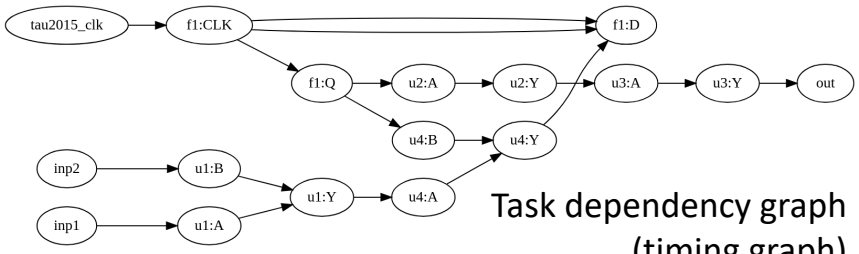
- **Measured the "pure" tasking performance**
  - Wavefront computing (regular compute pattern)
  - Graph traversal (irregular compute pattern)
  - Compared with OpenMP 4.5 and Intel TBB FlowGraph
    - G++ v8 with –fomp –O2 –std=c++17
    - Evaluated on a 4-core AMD CPU machine



Cpp-Taskflow scales the best when task counts (problem size) increases, using the least amount of code

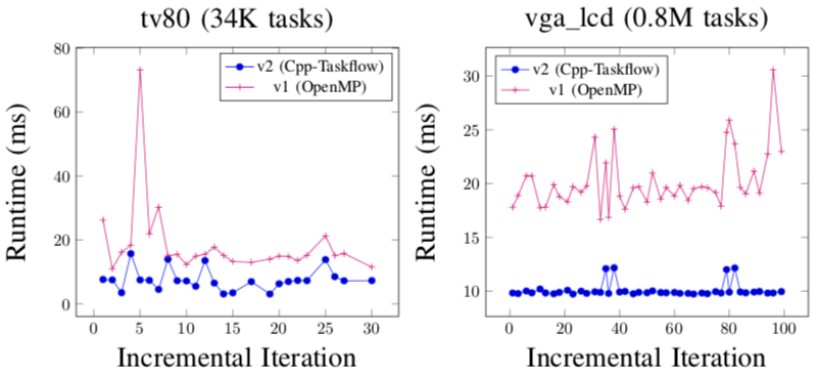
Large-Scale VLSI Timing Analysis

- **OpenTimer v1: A VLSI Static Timing Analysis Tool**
  - v1 first released in 2015 (open-source under GPL)
  - Loop-based parallelism using OpenMP 4.0
- **OpenTimer v2: A New Parallel Incremental Timer**
  - v2 first released in 2018 (open-source under MIT)
  - Task-based parallel decomposition using Cpp-Taskflow



Task dependency graph (timing graph)

Cost to develop is $275K with OpenMP vs $130K with Cpp-Taskflow! (https://dwheeler.com/sloccount/)



v2 (Cpp-Taskflow) is 1.4-2x faster than v1 (OpenMP)

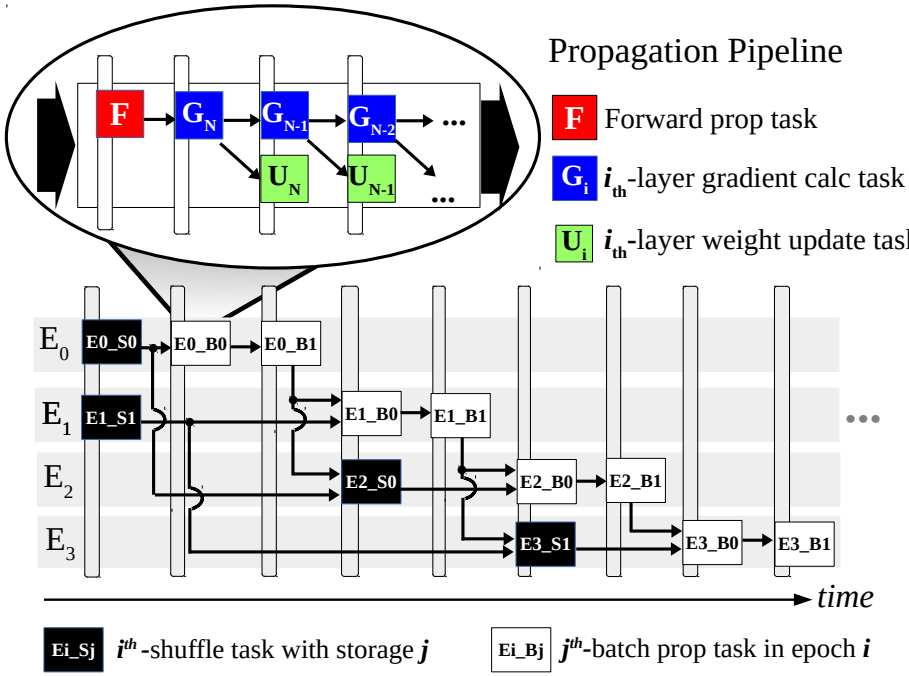
Deep Learning Model Training

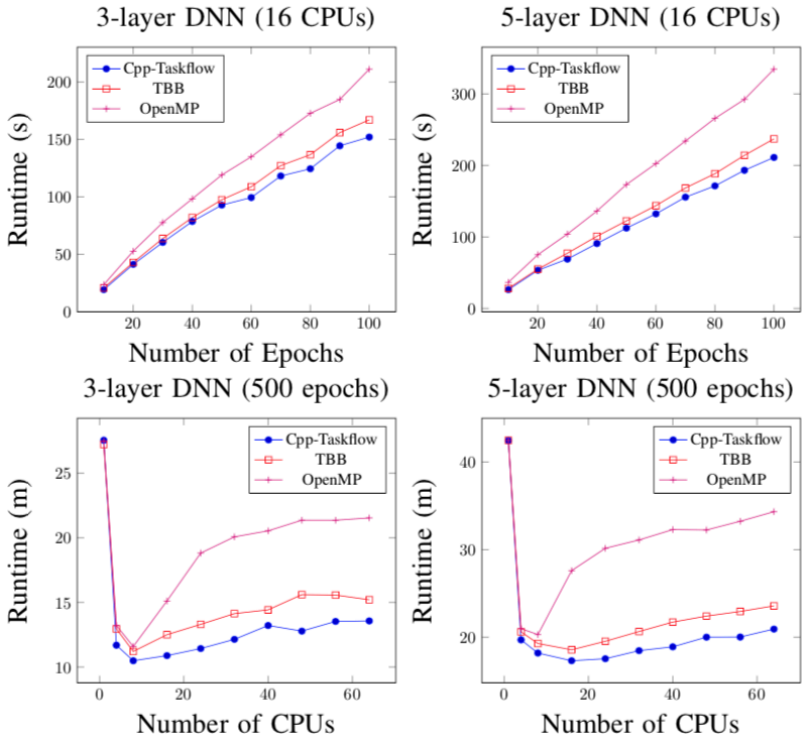
❑ 3-layer DNN and 5-layer DNN image classifier

Cpp-Taskflow			OpenMP			TBB			Sequential		
LOC	CC	T	LOC	CC	T	LOC	CC	T	LOC	CC	T
59	11	3	162	23	9	90	12	3	33	9	2

CC: cyclomatic complexity of the implementation
T: development time (in hours) by an experienced programmer

Dev time (hrs): 3 (Cpp-Taskflow) vs 9 (OpenMP)

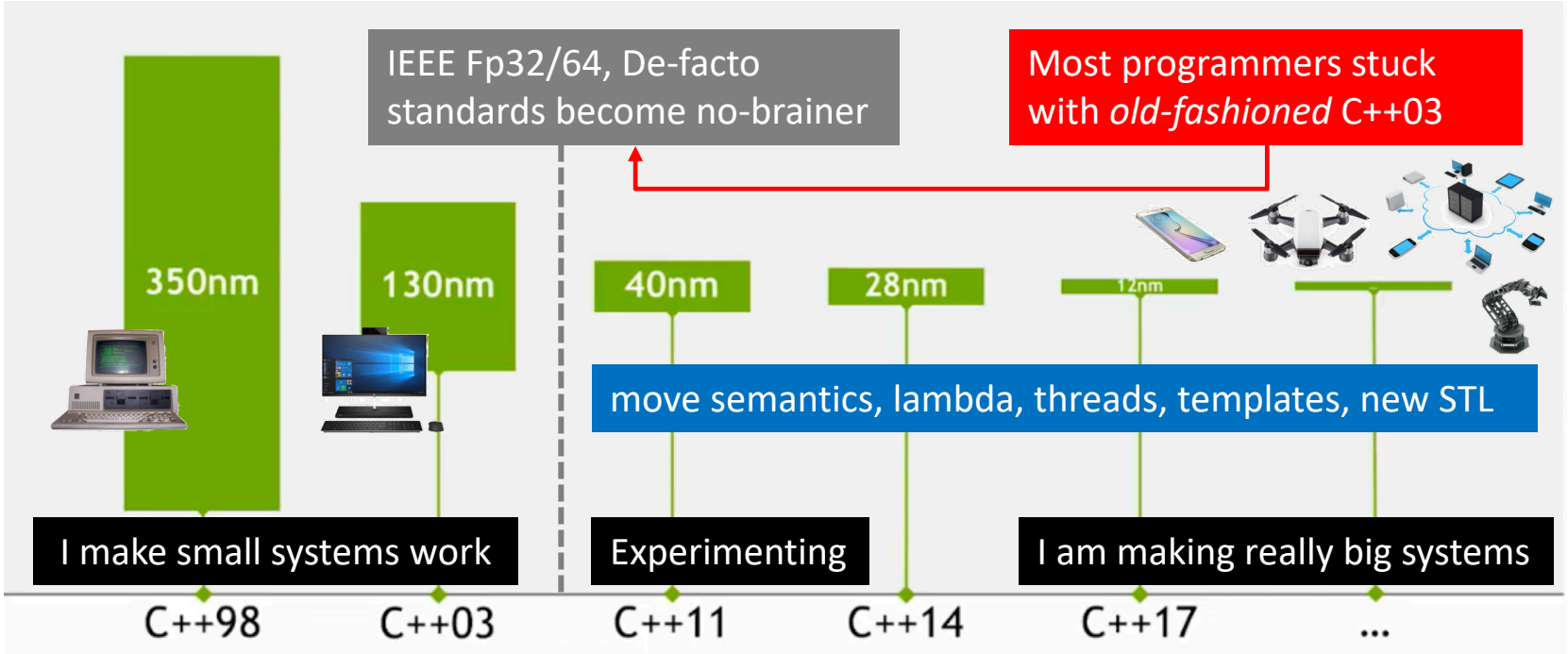




Cpp-Taskflow is about 10%-17% faster than OpenMP and Intel TBB in avg, using the least amount of source code

“Modern C++” Enables New Technology

- If you were able to tape out C++ …



- It’s much more than just being modern
  - Must “rethink” the way we used to design a program
  - Achieve the performance previously not possible

Modern programming languages allow us to achieve the performance scalability and programming productivity that were previously out of reach

What Cpp-Taskflow Users Say …

*"**Cpp-Taskflow is the cleanest Task API I have ever seen,**"* *Damien Hocking*

*"**Cpp-Taskflow has a very simple and elegant tasking interface; the performance also scales very well,**" Totalgee*

*"**Best poster award for open-source parallel programming library,**" Cpp-Conference (voted by 1000+ developers)*

People are using Cpp-Taskflow to speed up TensorFlow's kernel!

lifeiteng commented on Dec 11, 2018

Is your feature request related to a problem? Please describe.
I want to use TaskFlow in a C++ project which depends on TensorFlow. When compiling the Op & Kernel with `c++17`, there is a problem tensorflow/tensorflow#23561

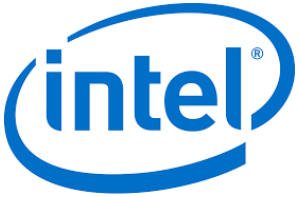
Outline

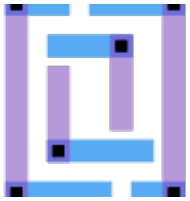
- **My early research at UIUC**
  - OpenTimer: A VLSI timing analysis tool
  - Express parallelism in the right way
- **A general-purpose distributed programming system**
  - DtCraft and its ecosystem
  - Machine learning, graph algorithms, chip design
  - Cpp-Taskflow: Modern C++ parallel task programming
- **Conclusion and future work**

Conclusion

- **Parallel task programming systems**
  - DtCraft: distributed computing
  - Cpp-Taskflow: multicore task programming
  - Solution at programming level matters a lot
    - Performance is top priority but never underestimate productivity
- **Three takeaways**
  - Need high-level abstractions for parallel programming
  - Need descent programming productivity
  - Need modern software technologies
- **CE students have unique strength in SE**
  - We understand both hardware and software
  - We understand both science and engineering

Acknowledgment (Users & Sponsors)





https://tsung-wei-huang.github.io/

Please contact me if you are interested in doing research for building software to solve real-world computer engineering problems