THE UNIVERSITY
of EDINBURGH

Hardware just-in-time compilation

Kimberley Stonehouse, Tom Spink and Björn Franke

UK Systems, 18th April 2024

What is just-in-time (JIT) compilation?

A brief history of just-in-time

- **Manufacturing** philosophy pioneered by Toyota in 1950s Japan

A brief history of just-in-time

- **Manufacturing** philosophy pioneered by Toyota in 1950s Japan
  - Creates items on demand, not in advance

"Making only what is needed, only when it is needed, and only in the amount that is needed" [1]

[1] Ohno, T., 1998. Toyota production system: beyond large-scale production. Productivity Press. ISBN 978-0-915299-14-0.

A brief history of just-in-time

- **Manufacturing** philosophy pioneered by Toyota in 1950s Japan
  - Creates items on demand, not in advance
  - Efficiency is now crucial to servicing demand

"Making only what is needed, only when it is needed, and only in the amount that is needed" [1]

[1] Ohno, T., 1998. Toyota production system: beyond large-scale production. Productivity Press. ISBN 978-0-915299-14-0.

A brief history of just-in-time

- **Manufacturing** philosophy pioneered by Toyota in 1950s Japan
  - Creates items on demand, not in advance
  - Efficiency is now crucial to servicing demand
  - Reduces resource usage and inventory cost

"Making only what is needed, only when it is needed, and only in the amount that is needed" [1]

[1] Ohno, T., 1998. Toyota production system: beyond large-scale production. Productivity Press. ISBN 978-0-915299-14-0.

A brief history of just-in-time

- The same philosophy can be applied to **compilation**

A brief history of just-in-time

- The same philosophy can be applied to **compilation**
  - Compiles functions on demand when called

A brief history of just-in-time

- The same philosophy can be applied to **compilation**
    - Compiles functions on demand when called
    - Reduces initial latency and memory usage

Compiling only what is needed, only when it is needed, and only in the amount that is needed

What are the benefits of just-in-time?

Benefits of just-in-time

- **Portability**
  - Responding to platform demand, rather than anticipating it

Benefits of just-in-time

- **Portability**
  - Responding to platform demand, rather than anticipating it
- **Optimisations**
  - Leveraging dynamic information to improve code quality

Benefits of just-in-time

- **Portability**
  - Responding to platform demand, rather than anticipating it
- **Optimisations**
  - Leveraging dynamic information to improve code quality
- **Performance**
  - Accelerating Java, Python, C#, PHP, JavaScript, Ruby, WebAssembly…

What challenges does just-in-time face?

Challenges faced by just-in-time

- Once again, efficiency is **crucial**
    - Latency ultimately results in program pauses

Challenges faced by just-in-time

- Once again, efficiency is **crucial**
  - Latency ultimately results in program pauses

The primary constraint on just-in-time compilers is speed. The program must not pause during execution.

How is latency mitigated?

Dynamic recompilation

- **Hot swapping**
  - Most programs spend the majority of time on a minority of code [1]

[1] Knuth, D.E., 1971. An empirical study of FORTRAN programs. Software: Practice and experience, 1(2), pp.105-133.

Dynamic recompilation

- **Hot swapping**
  - Most programs spend the majority of time on a minority of code [1]
  - Recompilation can yield significant performance improvements

[1] Knuth, D.E., 1971. An empirical study of FORTRAN programs. Software: Practice and experience, 1(2), pp.105-133.

Dynamic recompilation

- **Hot swapping**
  - Most programs spend the majority of time on a minority of code [1]
  - Recompilation can yield significant performance improvements
- **Tiered compilation**
  - Each tier delivers higher quality code, taking longer to do so

[1] Knuth, D.E., 1971. An empirical study of FORTRAN programs. Software: Practice and experience, 1(2), pp.105-133.

Dynamic recompilation

- **Hot swapping**
  - Most programs spend the majority of time on a minority of code [1]
  - Recompilation can yield significant performance improvements
- **Tiered compilation**
  - Each tier delivers higher quality code, taking longer to do so
  - Low latency early tiers conceal background optimisations

[1] Knuth, D.E., 1971. An empirical study of FORTRAN programs. Software: Practice and experience, 1(2), pp.105-133.

Introducing hardware

- **Hardware acceleration**
    - Adding hardware accelerated instructions can improve the efficiency of code generation by an average of 15% [1, 2]

[1] Carbon, M.A. et al., 2013. Hardware acceleration for just-in-time compilation on heterogeneous embedded systems. IEEE ASAP'13.
[2] Carbon, M.A. et al., 2014. Hardware acceleration of red-black tree management and application to just-in-time compilation. Journal of Signal Processing Systems.

Introducing hardware

- **Hardware acceleration**
  - Adding hardware accelerated instructions can improve the efficiency of code generation by an average of 15% [1, 2]

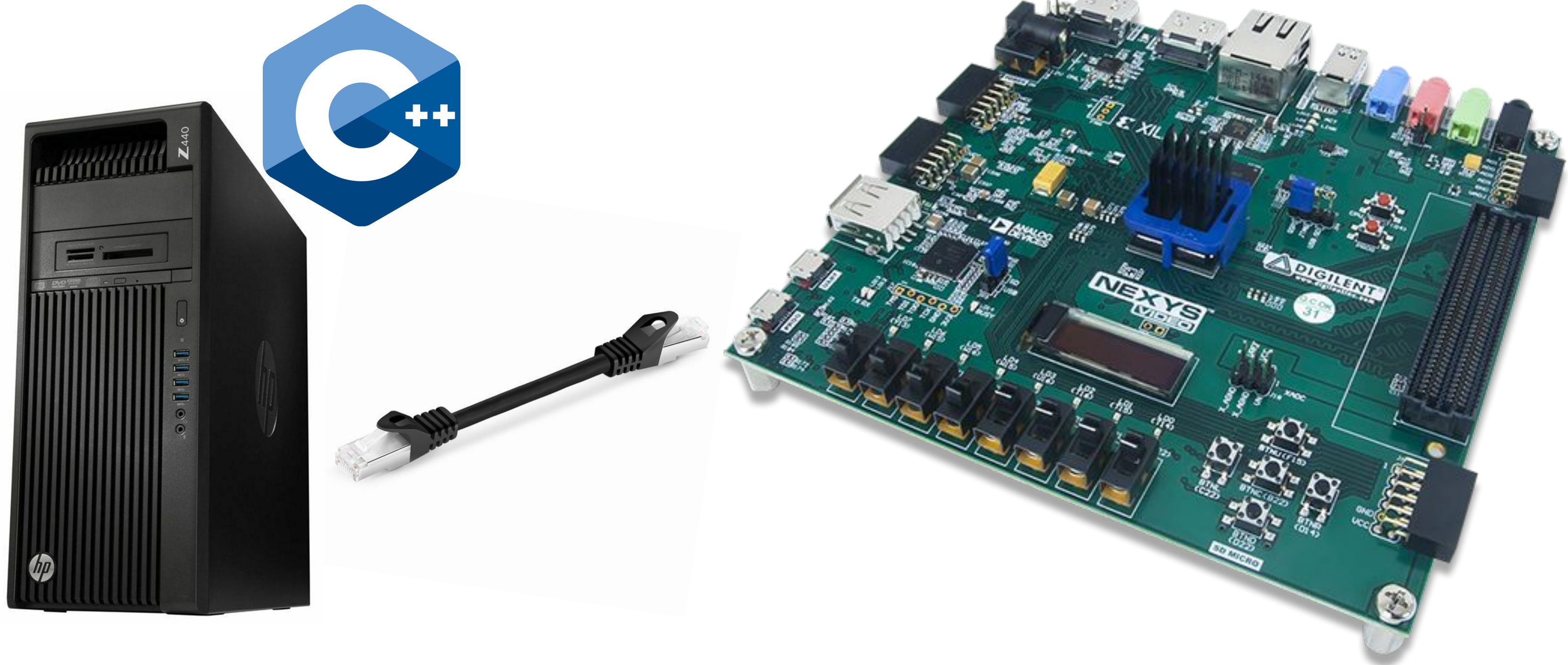
Can we combine tiered compilation with hardware acceleration?

[1] Carbon, M.A. et al., 2013. Hardware acceleration for just-in-time compilation on heterogeneous embedded systems. IEEE ASAP'13.
[2] Carbon, M.A. et al., 2014. Hardware acceleration of red-black tree management and application to just-in-time compilation. Journal of Signal Processing Systems.

What about a first tier entirely in hardware?

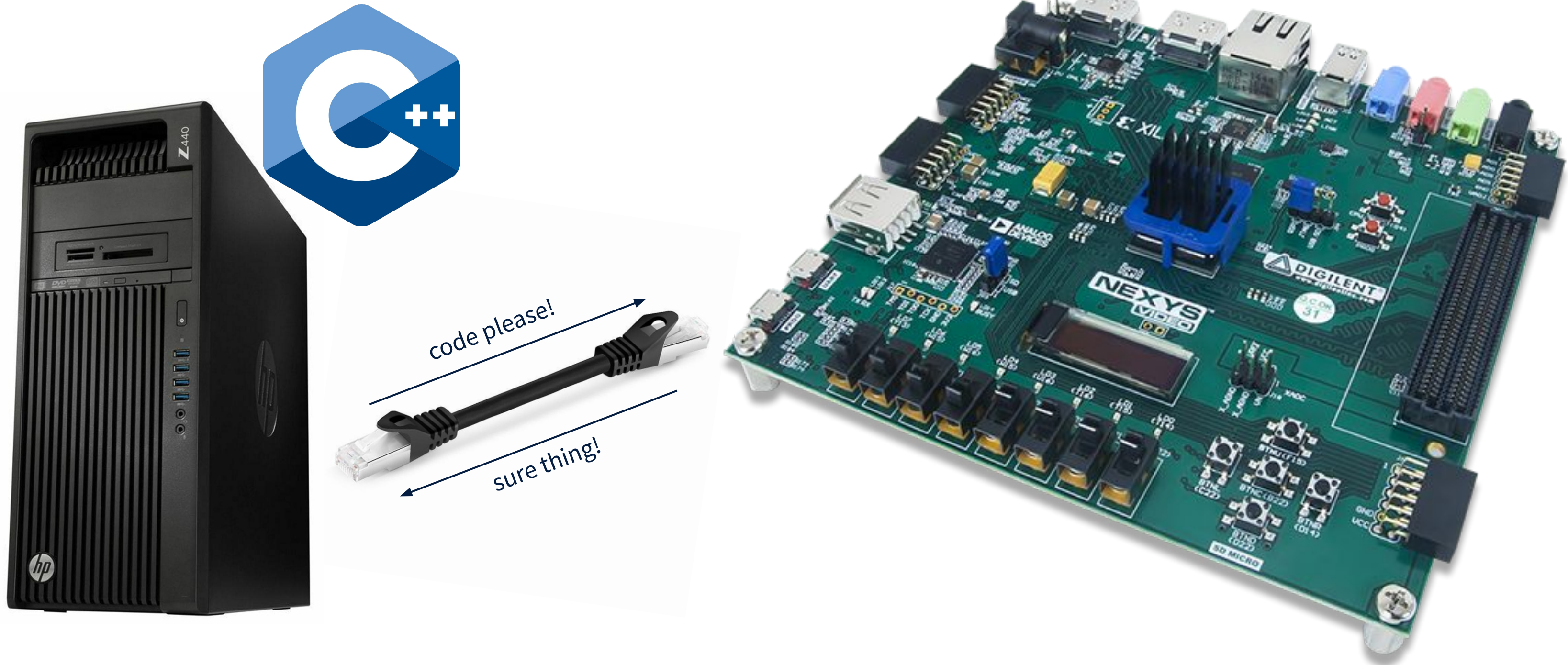
System architecture

System architecture



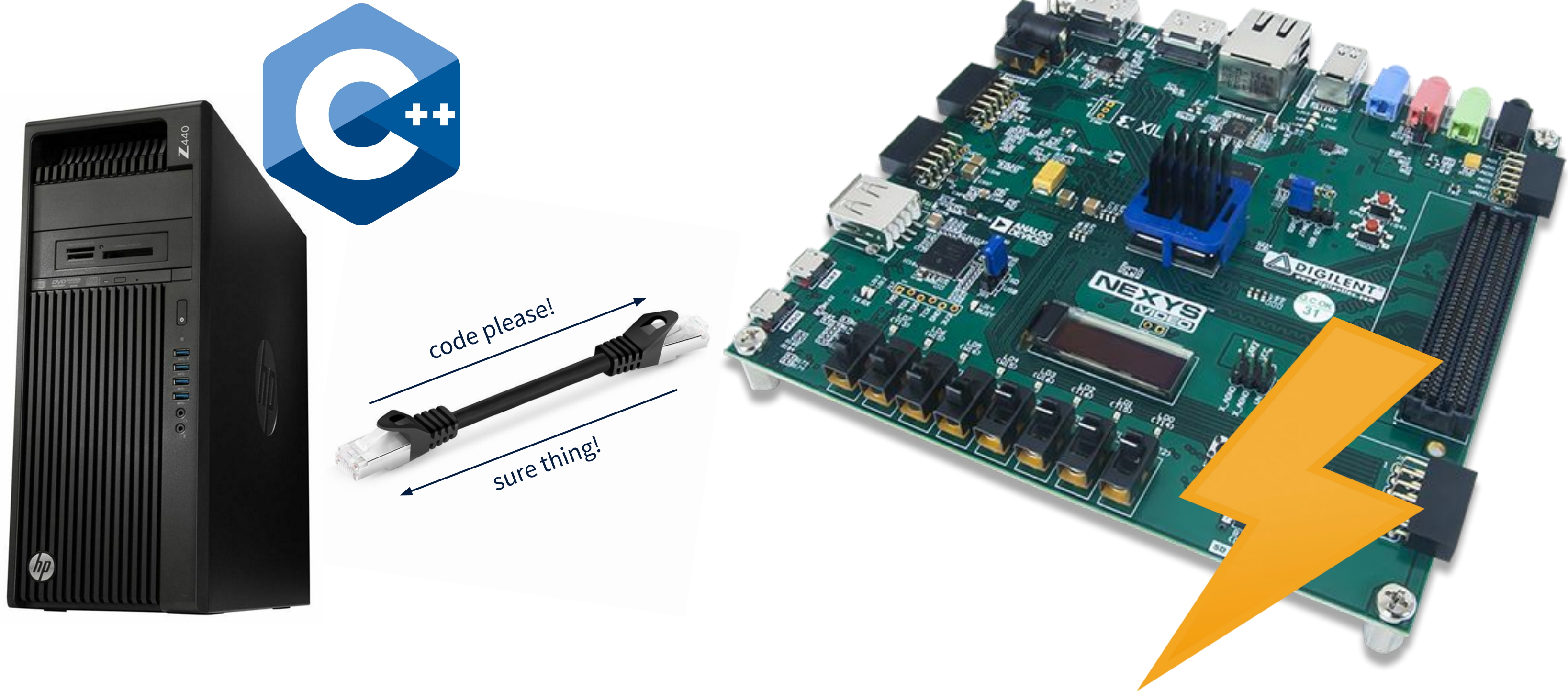
(not to scale)

System architecture

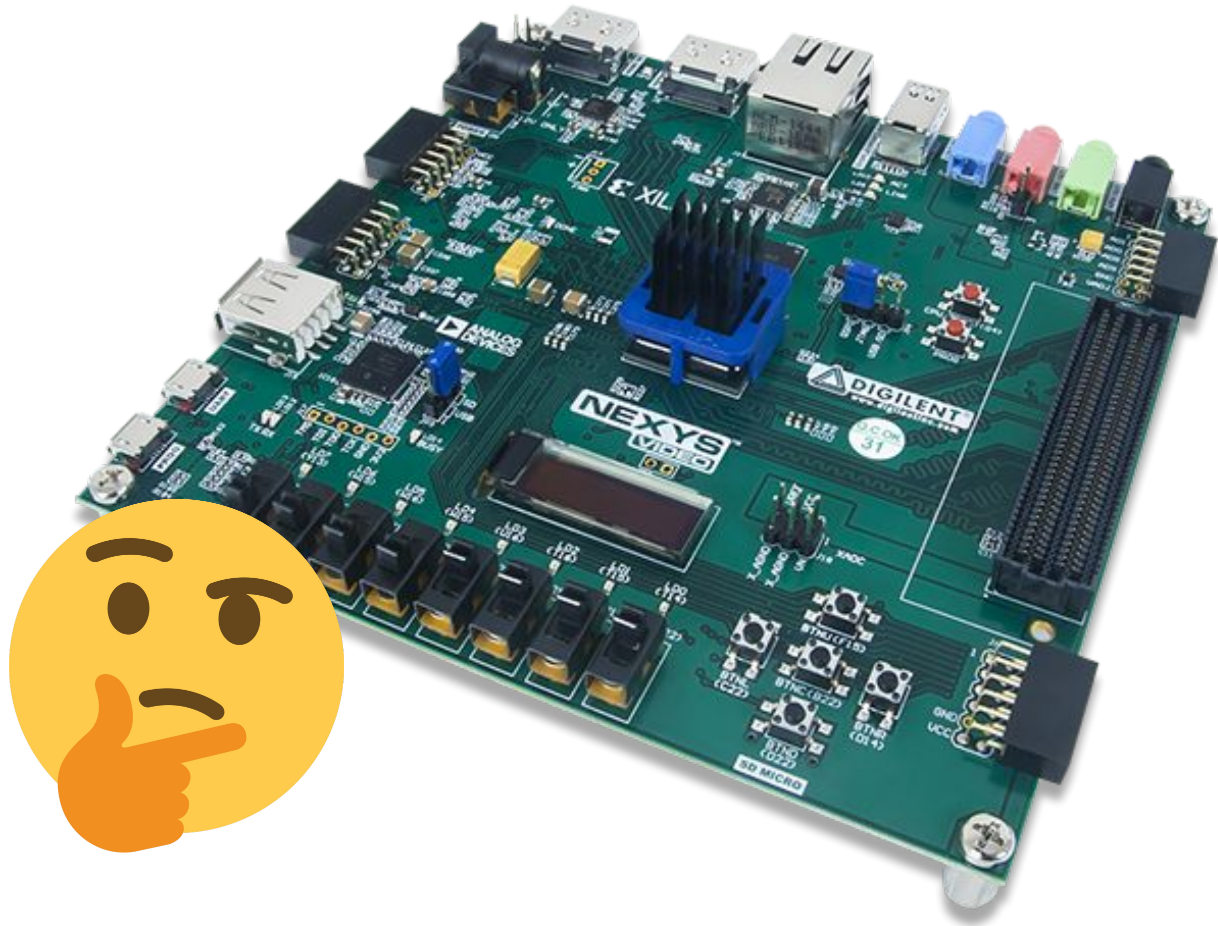


(not to scale)

System architecture



(not to scale)



(very not to scale)

Can we use high-level synthesis (HLS)?

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device
  - Code must conform to synthesisability constraints [1]

[1] AMD High-Level Synthesis User Guide.

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device
  - Code must conform to synthesisability constraints [1]
    - No system calls or dynamic memory management

[1] AMD High-Level Synthesis User Guide.

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device
  - Code must conform to synthesisability constraints [1]
    - No system calls or dynamic memory management
    - No recursion, virtual functions or variable length arrays

[1] AMD High-Level Synthesis User Guide.

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
    - Synthesise onto a programmable logic device
    - Code must conform to synthesisability constraints [1]
        - No system calls or dynamic memory management
        - No recursion, virtual functions or variable length arrays
        - No double pointers, function pointers or general pointer casting

[1] AMD High-Level Synthesis User Guide.

High-level synthesis (HLS)

- Develop a **software** prototype in C/C++
  - Synthesise onto a programmable logic device
  - Code must conform to synthesisability constraints [1]
    - No system calls or dynamic memory management
    - No recursion, virtual functions or variable length arrays
    - No double pointers, function pointers or general pointer casting
    - No library calls or standard template library containers

[1] AMD High-Level Synthesis User Guide.

Templating compiler

- Define a target **template** for each source instruction
  - At runtime, patch the template with dynamic values and emit

Templating compiler

- Define a target **template** for each source instruction
    - At runtime, patch the template with dynamic values and emit
    - Produces largely unoptimised code, but very quickly

Templating compiler

- Define a target **template** for each source instruction
  - At runtime, patch the template with dynamic values and emit
  - Produces largely unoptimised code, but very quickly
  - Simple implementation, so amenable to synthesis

Templating compiler

- Define a target **template** for each source instruction
  - At runtime, patch the template with dynamic values and emit
  - Produces largely unoptimised code, but very quickly
  - Simple implementation, so amenable to synthesis
- Demonstrated **benefits** in multiple contexts
  - Lightweight compilation in constrained environments [1]

[1] Coffin, E. et al., 2020. MicroJIT: a case for templated just-in-time compilation in constrained environments. CASCON'20.
[2] Kaur, H. et al., 2023. Performance Evaluation of Template-based JIT Compilation in OpenJ9. CASCON'23.
[3] Xu, H. et al., 2021. Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode. OOPSLA'21.

Templating compiler

- Define a target **template** for each source instruction
  - At runtime, patch the template with dynamic values and emit
  - Produces largely unoptimised code, but very quickly
  - Simple implementation, so amenable to synthesis
- Demonstrated **benefits** in multiple contexts
  - Lightweight compilation in constrained environments [1]
  - Reduced start up latency for larger virtual machines [2, 3]

[1] Coffin, E. et al., 2020. MicroJIT: a case for templated just-in-time compilation in constrained environments. CASCON'20.

[2] Kaur, H. et al., 2023. Performance Evaluation of Template-based JIT Compilation in OpenJ9. CASCON'23.

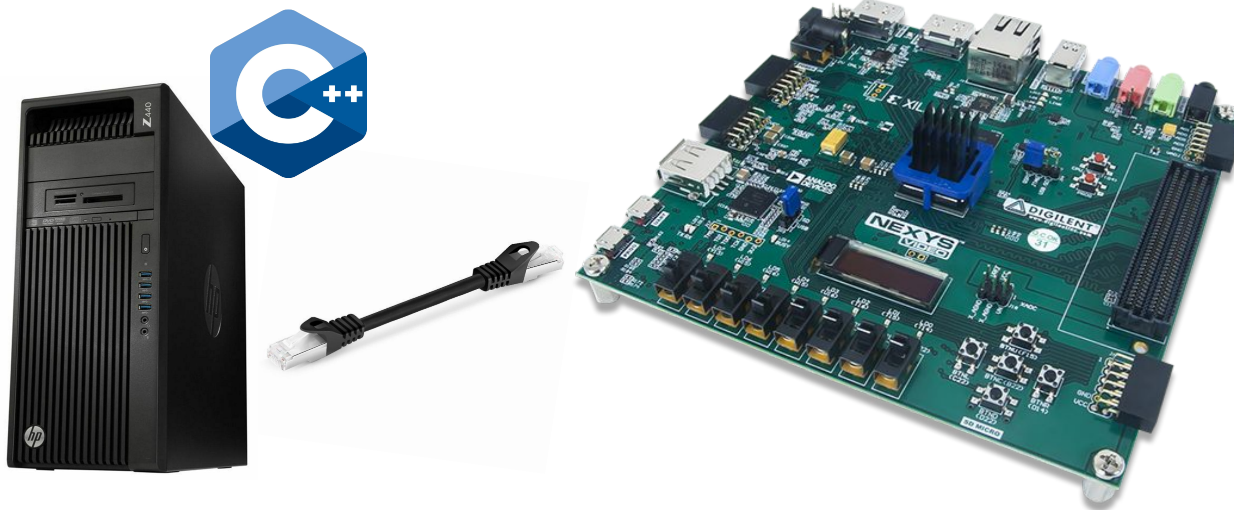
[3] Xu, H. et al., 2021. Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode. OOPSLA'21.

Current progress

- **Proof of concept** implementation
  - Translates register-based intermediate representation to native x86

Current progress

- **Proof of concept** implementation
  - Translates register-based intermediate representation to native x86
  - Transfers code back and forth via physical Ethernet connection



Future work

- Extend **language** support
    - Use a portable bytecode like WebAssembly

Future work

- Extend **language** support
  - Use a portable bytecode like WebAssembly
  - Requires stack-based to register-based conversion

Future work

- Extend **language** support
  - Use a portable bytecode like WebAssembly
  - Requires stack-based to register-based conversion
- **Optimise** hardware implementation
  - Synthesisable design != optimal design

Future work

- Extend **language** support
  - Use a portable bytecode like WebAssembly
  - Requires stack-based to register-based conversion
- **Optimise** hardware implementation
  - Synthesisable design != optimal design
  - Requires hardware knowledge

Future work

- Extend **language** support
  - Use a portable bytecode like WebAssembly
  - Requires stack-based to register-based conversion
- **Optimise** hardware implementation
  - Synthesisable design != optimal design
  - Requires hardware knowledge
- Measure system **performance**
  - Choose appropriate benchmarks for evaluation

Summary

- **Just-in-time** compilation brings many benefits
  - Dynamic optimisations can be very powerful

Summary

- **Just-in-time** compilation brings many benefits
  - Dynamic optimisations can be very powerful
  - But speed is still the primary limiting factor

Summary

- **Just-in-time** compilation brings many benefits
  - Dynamic optimisations can be very powerful
  - But speed is still the primary limiting factor
- **Hardware** just-in-time is a promising space
  - Hardware acceleration has shown positive results

Summary

- **Just-in-time** compilation brings many benefits
  - Dynamic optimisations can be very powerful
  - But speed is still the primary limiting factor
- **Hardware** just-in-time is a promising space
  - Hardware acceleration has shown positive results
  - Research into an entirely hardware first tier is ongoing

Summary

- **Just-in-time** compilation brings many benefits
  - Dynamic optimisations can be very powerful
  - But speed is still the primary limiting factor
- **Hardware** just-in-time is a promising space
  - Hardware acceleration has shown positive results
  - Research into an entirely hardware first tier is ongoing
  - But there are still many interesting problems left to solve