Which framework to use?
A “working group” is on it

Portability: A Necessary Approach for Future Scientific Software
Meghana Bhattacharya, Paolo Calafiura, Taylor Childers, Mark Dewing, Zhihua Dong, Oliver Gutsche, Salman Habib, Xiangyang Ju, Michael Kirby, Kyle Knoepfel, Matti Kortelainen, Martin Kwok, Charles Leggett, Meifeng Lin, Vincent R. Pascuzzi, Alexei Strelchenko, Brett Viren, Beomki Yeo, Haiwang Yu

Today's world of scientific software for High Energy Physics (HEP) is powered by x86 code, while the future will be much more reliant on accelerators like GPUs and FPGAs. The portable parallelization strategies (PPS) project of the High Energy Physics Center for Computational Excellence (HEP/CCE) is investigating solutions for portability techniques that will allow the coding of an algorithm once, and the ability to execute it on a variety of hardware products from many vendors, especially including accelerators. We think without these solutions, the scientific success of our experiments and endeavors is in danger, as software development could be expert driven and costly to be able to run on available hardware infrastructure. We think the best solution for the community would be an extension to the C++ standard with a very low entry bar for users, supporting all hardware forms and vendors. We are very far from that ideal though. We argue that in the future, as a community, we need to request and work on portability solutions and strive to reach this ideal.

Question #1: Should we include such working group?
A “working group” is on it

Portability: A Necessary Approach for Future Scientific Software
Meghna Bhattacharya, Paolo Calafiura, Taylor Childers, Mark Dewing, Zhihua Dong, Oliver Gutsche, Salman Habib, Xiangyang Ju, Michael Kirby, Kyle Knoepfel, Matti Kortelainen, Martin Kwok, Charles Leggett, Meifeng Lin, Vincent R. Pascuzzi, Alexei Strelchenko, Brett Viren, Beomki Yeo, Haiwang Yu

Today's world of scientific software for High Energy Physics (HEP) is powered by x86 code, while the future will be much more reliant on accelerators like GPUs and FPGAs. The portable parallelization strategies (PPS) project of the High Energy Physics Center for Computational Excellence (HEP/CCE) is investigating solutions for portability techniques that will allow the coding of an algorithm once, and the ability to execute it on a variety of hardware products from many vendors, especially including accelerators. We think without these solutions, the scientific success of our experiments and endeavors is in danger, as software development could be expert driven and costly to be able to run on available hardware infrastructure. We think the best solution for the community would be an extension to the C++ standard with a very low entry bar for users, supporting all hardware forms and vendors. We are very far from that ideal though. We argue that in the future, as a community, we need to request and work on portability solutions and strive to reach this ideal.

Question #1: Should we include such working group?  
Question #2: Do they support std::par?
Portability: A Necessary Approach for Future Scientific Software
Meghna Bhattacharya, Paolo Calafiura, Taylor Childers, Mark Dewing, Zhihua Dong, Oliver Gutsche, Salman Habib, Xiangyang Ju, Michael Kirby, Kyle Knoepfel, Matti Kortelainen, Martin Kwok, Charles Leggett, Meifeng Lin, Vincent R. Pascuzzi, Alexei Strelchenko, Brett Viren, Beomki Yeo, Haiwang Yu

Today's world of scientific software for High Energy Physics (HEP) is powered by x86 code, while the future will be much more reliant on accelerators like GPUs and FPGAs. The portable parallelization strategies (PPS) project of the High Energy Physics Center for Computational Excellence (HEP/CCE) is investigating solutions for portability techniques that will allow the coding of an algorithm once, and the ability to execute it on a variety of hardware products from many vendors, especially including accelerators. We think without these solutions, the scientific success of our experiments and endeavors is in danger, as software development could be expert driven and costly to be able to run on available hardware infrastructure. We think the best solution for the community would be an extension to the C++ standard with a very low entry bar for users, supporting all hardware forms and vendors. We are very far from that ideal though. We argue that in the future, as a community, we need to request and work on portability solutions and strive to reach this ideal.

Question #1: Should we include such working group?
Question #2: Do they support std::par?
Question #3: Not sure to understand that statement...
1. **Ease of Learning**
   - Assess the ease of learning for developers who are familiar with CUDA or other GPU language and for novices.
   - How are domain experts/novices typically contributing to the development?
     - e.g. Who is porting code, who is writing new algorithms.
     - Domain experts who know the code intimately and know which parts are costly and which are not.
     - Novices (grad students and postdocs) who get the task to optimize parts (kernels) of the code under the instruction of domain experts.

2. **Code conversion**
   - From CPU code to GPU code.
   - From low-level GPU code (CUDA, etc.) to higher level portability code.
   - From one portability framework to another.

3. **Extent of modifications to existing code**
   - Control of main().
   - Threading/execution model.

4. **Extent of modification to Event Data Model (EDM)**
   - How are data transfer handled across different memory space?
   - How are data access handled across different memory space?

5. **Extent of modification to build system**
   - How much code needs to be recompiled.
   - Cmake or make changes/integration.

6. **Hardware Mapping**
   - Current and promised future support of various hardware platforms.
   - Will all these technologies work on all the current and future platforms?
   - When can we expect to have implementations?

7. **Feature Availability**
   - Reductions, kernel chaining, callbacks, etc.
   - Concurrent kernels.
   - Support for interfacing to optimized math-heavy libraries by the technology across different hardware platforms.
     - How easy is it to use these libraries within the portability layer?
     - Use cases include random number generators, FFT.

8. **Address of needs of all workflows**
   - Scaling with number of kernels per application (LHC has many, Neutrino has few).
   - Do you require support infrastructure within collaboration?
   - How well does the support scale with the number of collaborators?
   - How well do the developers of the portability layer support the users?
     - Channel for providing support (e.g. Slack/private communication etc).
     - Responsiveness.

9. **Long term sustainability and code stability**
   - Support model of technologies → stability of implementation if underlying libraries (CUDA) change.
   - CUDA is going to be around for a long time, what about the portability solutions?
   - Long term support for technologies by vendors.

10. **Compilation time**
    - How deep will the technology penetrate the code base (down to the EDM) and extend complication time?
    - Do we need one build for everything or do we need different compilations for different platforms (CPU, AMD GPU, Intel GPU, NVidia GPU, ...)?
    - Compatibility with experiment's software distribution strategies (we need to use local, optimized libraries for specific HPCs).

11. **Run time**
    - Running same use case example on CPU w/ new design on accelerator on comparable resources.
    - Does it degrade performance of CPU code (or use significantly more memory).
    - “Are you memory bound?” “Are you CPU bound?”

12. **Ease of debugging**
    - How easy is it to debug implementations of code in the technologies.

13. **Aesthetics**
    - What's aesthetic to one may be ugly to another.
    - Compatibility with general C++ code philosophy.
    - Compatibility with evolving standards in the experiments code base (C++17 compatibility, etc.)

14. **Interoperability**
    - Run different technologies in the same application?
      - How are externals treated? (CMSSW is using Kokkos, but Geant is using Alpaka, ...)
    - Interaction with existing thread pool on CPU/CPU backends?
    - How easy is it to switch from e.g. TBB to HPX?
    - Can the portability layer be used together with the native API of a particular platform within an application? E.g. in order to have part of the algorithm more optimized.
Metric to decide...

1. **Ease of Learning**
   - Assess the ease of learning for developers who are familiar with CUDA or other GPU language and for novices
   - How are domain experts/novices typically contributing to the development?
   - E.g., Who is porting code, who is writing new algorithms
   - Domain experts who know the code intimately and know which parts are costly and which are not
   - Novices (grad students and postdocs) who get the task to optimize parts (kernels) of the code under the instruction of domain experts

2. **Code conversion**
   - From CPU code to GPU code
   - From low-level CPU code (CUDA, etc.) to higher-level portability code
   - From one portability framework to another

3. **Extent of modifications to existing code**
   - Control of main
   - Threading/execution model

4. **Extent of modification to Event Data Model (EDM)**
   - How are data transfer handled across different memory space?
   - How are data access handled across different memory space?

5. **Extent of modification to build system**
   - How much code needs to be recompiled
   - Cmake or make changes/integration

6. **Hardware Mapping**
   - Current and promised future support of various hardware platforms
   - Will all these technologies work on all the current and future platforms?
   - When can we expect to have implementations?

7. **Feature Availability**
   - Reductions, kernel chaining, callbacks, etc.
   - Concurrent kernels
   - Support for interfacing to optimized math-heavy libraries by the technology across different hardware platforms?
   - How easy is it to use these libraries within the portability layer?
   - Use cases include random number generators, FFT?

8. **Address needs of all workflows**
   - Scaling with number of kernels per application (LHC has many, Neutrino has few)
   - Do you require support infrastructure within collaboration?
   - How well do the support scale with the number of collaborators?
   - How well do the developers of the portability layer support the users?
   - Channel for providing support (e.g., Slack/private communication etc.)
   - Responsiveness

9. **Long term sustainability and code stability**
   - Support model of technologies → stability of implementation if underlying libraries (CUDA) change
   - CUDA is going to be around for a long time, what about the portability solutions?
   - Long term support for technologies by vendors

10. **Compilation time**
    - How deep will the technology penetrate the code base (down to the EDM) and extend compilation time?
    - Do we need one build for everything or do we need different compilations for different platforms (CPU, AMD GPU, Intel GPU, NVIDIA GPU, …)
    - Compatibility with experiment’s software distribution strategies (we need to use local, optimized libraries for specific HPCs)

11. **Run time**
    - Running same use case example on CPU w/ new design on accelerator on comparable resources
    - Does it degrade performance of CPU code (or use significantly more memory)
    - “Are you memory bound”? “Are you CPU bound?”

12. **Ease of debugging**
    - How easy is it to debug implementations of code in the technologies

13. **Aesthetics**
    - What’s aesthetic to one may be ugly to another
    - Compatibility with general C++ code philosophy
    - Compatibility with evolving standards in the experiments code base (C++17 compatibility, etc.)

14. **Interoperability**
    - Run different technologies in the same application?
    - How are externals treated? (CMSSW is using Kokkos, but Geant is using Alpaka, …)
    - Interaction with existing thread pool on CPU/CPU backends?
    - How easy is it to switch from e.g., TBB to HPX?
    - Can the portability layer be used together with the native API of a particular platform within an application? E.g. in order to have part of the algorithm more optimized
Other code/group strategy

CMS is pushing for alpaca.
Atlas, I do not know.
LHAPDF -> kokkos (experimental and open to change).
Sherpa -> seems CUDA
PDFFLOW -> python/tensorflow
Particle track -> (2002.11529) CUDA
**My conclusion**

The decision will be taken by the community in multiple years from now.

The recommendation of the paper is too weak.

My number 1 criteria is Long term sustainability

-> CUDA / HIP / std:par