



Deposited via The University of York.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/136574/>

Version: Accepted Version

Conference or Workshop Item:

Truby, David R., Bertolli, Carlo, Wright, Steven A. et al. (Accepted: 2018) Implicit mapping of pointers inside C++ Lambda closure objects in OpenMP target offload regions. In: UK OpenMP Users' Conference 2018, 21-22 May 2018. (In Press)

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Implicit Mapping of Pointers Inside C++ Lambda Closure Objects in OpenMP Target Offload Regions

David Truby¹, Carlo Bertolli², Steven Wright¹, Gheorghe-Teodor Bercea², Kevin O'Brien², and Stephen Jarvis¹

¹University of Warwick

²IBM Research

February 28, 2018

Abstract

With the diversification of HPC architectures beyond traditional CPU-based clusters, a number of new frameworks for performance portability across architectures have arisen. One way of implementing such frameworks is to use C++ templates and lambda expressions to design loop-like functions. However, lower level programming APIs that these implementations must use are often designed with C in mind and do not specify how they interact with C++ features such as lambda expressions.

This paper proposes a change to the behavior of the OpenMP specification with respect to lambda expressions such that when functions generated by lambda expressions are called inside GPU regions, any pointers used in the lambda expression correctly refer to device pointers. This change has been implemented in a branch of the Clang C++ compiler and demonstrated with two representative codes. Our results show that the implicit mapping of lambda expressions always exhibits identical performance to an explicit mapping but without breaking the abstraction provided by the high level frameworks, and therefore also reduces the burden on the application developer.