

SANDIA REPORT

SAND2012-5040
Unlimited Release
Printed June 2012

Dual Compile Strategy for Parallel Heterogeneous Execution

Tyler B. Smith and James T. Perry

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



SAND2012-5040
Unlimited Release
Printed June 2012

Dual Compile Strategy for Parallel Heterogeneous Execution

Tyler B. Smith and James T. Perry
Surety Electronics and Software
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0451

Abstract

The purpose of the *Dual Compile Strategy* is to increase our trust in the *Compute Engine* during its execution of instructions. This is accomplished by introducing a heterogeneous *Monitor Engine* that checks the execution of the *Compute Engine*. This leads to the production of a second and custom set of instructions designed for monitoring the execution of the *Compute Engine* at runtime. This use of multiple engines differs from redundancy in that one engine is working on the application while the other engine is monitoring and checking in parallel instead of both applications (and engines) performing the same work at the same time.

DUAL COMPILE STRATEGY FOR PARALLEL HETEROGENEOUS EXECUTION

Figure 1 shows the *Dual Compile Strategy for Parallel Heterogeneous Execution*. The purpose of the *Dual Compile Strategy* is to increase our trust in the *Compute Engine* during its execution of instructions. This is accomplished by introducing a *Monitor Engine* that checks the execution of the first engine.

This leads to the production of a custom set of instructions designed for monitoring the execution of the *Compute Engine* at runtime. This new set of monitor instructions could be developed in two different ways. First, compile the monitor instructions from the application's source. Second, compile the monitor instructions from the computation compiler's output. This concept is what is called a *Dual Compile Strategy* as shown in Figure 1.

Architectural Benefits

This use of multiple engines differs from redundancy and diverse redundancy (e.g. triple modular redundancy in hardware and N-version programming in software) in that one engine is working on the application while the other engine is monitoring and checking in parallel instead of both applications (and engines) performing the same work at the same time. Since both engines are working on completely different (yet related) tasks, it would be extremely difficult for both instruction streams to be modified and remain valid.

One aspect of this concept is similar to self-checking code. However, self-checking requires additional work of the *Compute Engine* which results in reduced performance whereas our separate *Monitor Engine* works in parallel to the *Compute Engine*, allowing it to maintain performance. Additionally, the *Monitor Engine* goes beyond simply self-checking code by verifying the interaction of the *Compute Engine* with internal and external hardware components.

Ideally all components in the system would be verified and trusted, such as the application source code, both compilers, and both engines. Due to the complexity of the computation path, it may not be feasible to rigorously verify its compiler and engine. However, if the monitor path is simpler, it might be feasible to verify its compiler and engine to boost the trust of the system.

Besides the potential trust and security benefits, there is also potential for improved reliability by leveraging this architecture to implement fault tolerance.

Monitor and Check Description

Instruction Stream Checks

These checks done by the *Monitor Engine* are based off of the *Monitor Instruction Stream* which would be designed to validate a corresponding *Computation Instruction Stream*. Consider a one-to-one correlation between each instruction of the two streams.

For example, assume a relative jump instruction (*relativeJump*) does the following:

1. Fetches an instruction argument from the program memory at PC
2. Stores the sum of PC with the argument into PC
3. Jumps to the address contained in PC

A corresponding monitor instruction *relativeJump_check* would confirm the following events (and no other events) took place:

1. Register read: PC
2. Program memory read
3. Register write: operandRegister1
4. Register move: PC to operandRegister2
5. ALU operation: addition
6. Register write: resultRegister
7. Register move: resultRegister to PC
8. Branch signal assertion

In this way, the monitor's instructions correspond to the Operational Signature of the *Compute Engine*. To clarify, an 'Operational Signature' is a term coined herein to describe the set of operations and their proper order (the signature) for a given instruction as previously exemplified.

As the *Compute Engine* executes its set of instructions, the *Monitor Engine* executes the monitor instructions in parallel. This allows the monitor to perform runtime checks of the *Compute Engine*. Possible types of checks include, but are not limited to:

1. Specific memory reads/writes (program, data, ...)
2. Specific register reads/writes (PC, SP, ...)
3. Use of Arithmetic Logic Unit (ALU)
4. Assertion of control signals

Hardware checks

In addition to the monitoring of the Operational Signature, there are many properties which should remain true in nearly all instances (independent of a given operation). These checks would be done in addition to the previously mentioned monitor instructions:

1. Stack accesses always occur between the frame pointer and the stack pointer (except during the building and destruction of the stack frame)
2. Registers dedicated for constant values are never written to except during initialization

Offload Application checks

In some languages, there are many runtime-checks built into the language that the *Compute Engine* needs to perform during execution. These checks help to make the language less susceptible to many common problems, such as buffer-overflows (or other runtime exceptions). Since these checks are built-in, the computation throughput is reduced.

However, the *Dual Compile Strategy* could boost the efficiency of the computations by off-loading the runtime checks to the monitor. Since the monitor can do these checks in parallel, the *Compute Engine* would be free to continue with the important computations.

Monitor Limitations

Even with a long list of potential capabilities, there are still some limitations of the *Monitor Engine*. The more accurate of a check the monitor performs, the closer it mimics the computational path. A high level of accuracy obtained by computing the results to compare with the *Compute Engine* may result in an undesirable homogeneous solution.

For example, the previously mentioned *relativeJump_check* on step 5 merely checks that the adder of the ALU was used, not that the ALU computed the proper result. So, if we are confident that each individual instruction matched its Operational Signature, then we have higher confidence that the result of the arithmetic operation is valid and that the overall execution is behaving normally.

Due to the complex nature of compilers (especially compiler optimization), it can be difficult for the monitor compiler and compute compiler to produce instructions that can be synchronously executed. For this reason, it would be much simpler for the monitor compiler to produce the monitor instructions from the compute instructions, rather than directly from source. Truly independent compilation is ideal, since the dependent form of monitor compilation can only be as correct as the compute instructions it is given.

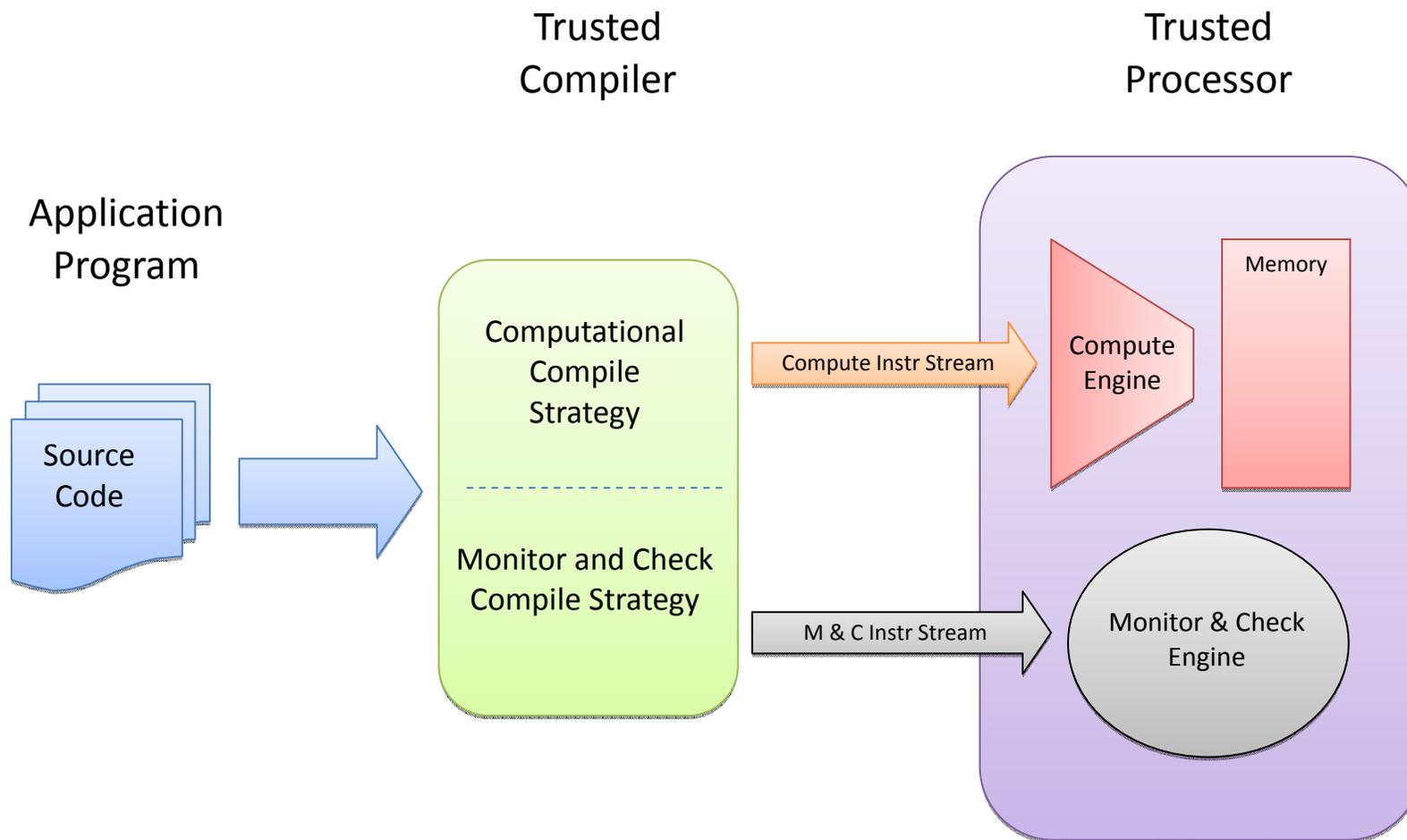


Figure 1: A Dual Compile Strategy for Parallel Heterogeneous Execution

ELECTRONIC DISTRIBUTION:

- 1 J. A. MCCOY
31 Camino de Avila
Tijeras, NM 87059

- 1 MS 0114 R. T. WESTERVELT, 01931
- 1 MS 0161 D. J. JENKINS, 11500
- 1 MS 0451 S. M. BECKER, 02144
- 1 MS 0451 T. B. SMITH, 02144
- 1 MS 0453 B. L. REMUND, 2140
- 1 MS 0487 A. L. HILLHOUSE, 02142
- 1 MS 0899 Technical Library, 09536
- 1 MS 0359 D. Chavez, LDRD Office, 01911

