# **TLS and Parallelism**

N3487=12-0177 Pablo Halpern, Intel Corp <u>pablo.g.halpern@intel.com</u> Presented to SG1 (Parallelism) 2012-05-08



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



# Abstract

I present three different realistic use cases for thread-local storage (TLS). Using examples in PPL, Cilk, and TBB, I show that no one definition of the thread local storage class would suffice for all three use cases, regardless of parallelism platform. We thus need to broaden our support for TLS, with new language and/or library features, in order to allow parallel programming to co-exist with the different uses of TLS. Finally, I propose a specific meaning for the existing keyword, thread local, matching one of the three use cases.



Software & Services Group Developer Products Division



# **Parallelism Terminology**

- **Task**: A unit of work that can be scheduled and executed asynchronously. Examples of tasks:
  - Each iteration of a parallel loop
  - The invokable argument of std::async
  - The branches of a parallel\_invoke in PPL or TBB
  - The continuation of a cilk\_spawn I.e., the code that runs between the cilk\_spawn and the corresponding cilk\_sync
- Worker: The member of a thread pool that executes a task. Worker threads are typically managed by a parallelism runtime library and are re-used many times for many tasks.



Software & Services Group Developer Products Division



# **Use case 1: Session-specific information**

### The Setup:

We are creating a web application which creates a new thread for each user session. Session information is stored in a thread-local variable:

```
struct session_info {
    int user_id;
    unsigned long long crypt_key[2];
    ...
};
thread local session info my session;
```



Software & Services Group Developer Products Division



### **Use case 1: Serial code**

```
void process(record& r) {
    decrypt(r, my_session.crypt_key);
    ...
                         thread-local lookup
void on submit()
    record shopping cart, order history;
    ...
    process(shopping_cart);
    process(order history);
```



Software & Services Group Developer Products Division



# **Use case 1: Parallelized with PPL**

```
void process(record& r) {
    decrypt(r, my_session.crypt key);
    ...
                       Want TLS bound to
                       user thread, not to
                         worker thread
void on submit()
    record shopping cart, order history;
    ...
    Concurrency::parallel_invoke(
        [&] { process(shopping cart); },
        [&] { process(order history); } );
```



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



# **Use case 1: Parallelized with Cilk**

```
void process(record& r) {
    decrypt(r, my session.crypt key);
    ...
                       Want TLS bound to
                       user thread, not to
                         worker thread
void on submit()
    record shopping cart, order history;
    ...
    cilk spawn process(shopping cart);
    process(order history);
```



Software & Services Group Developer Products Division



# **Use Case 2: Dynamic Cache**

### The Setup:

We wish to save computation time in a multithreaded application by caching previouslycomputed values in a hashed container. Threadlocal storage provides a (seemingly) easy way to implement such a cache without having to worry about synchronizing between threads. The occasional redundant computations caused by the lack of a shared cache add an acceptable cost for our data set.

#### thread\_local

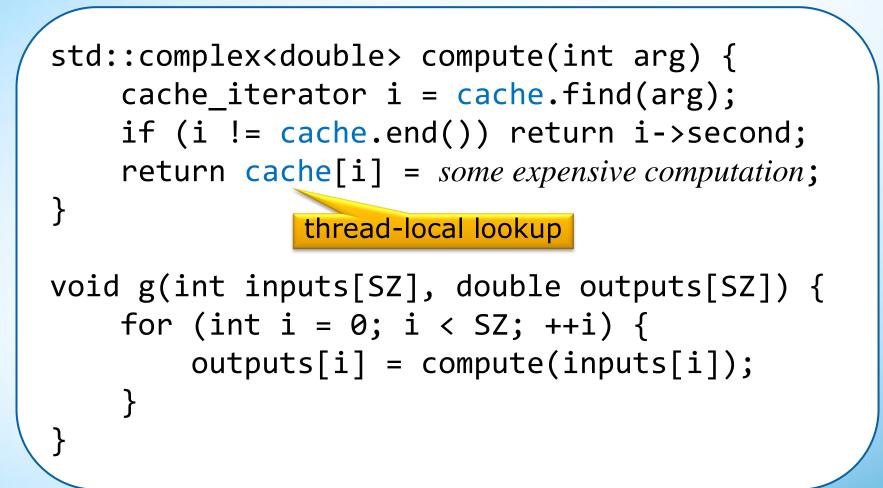
my\_cache\_class<int,complex<double>> cache;



Software & Services Group Developer Products Division



# **Use case 2: Serial code**





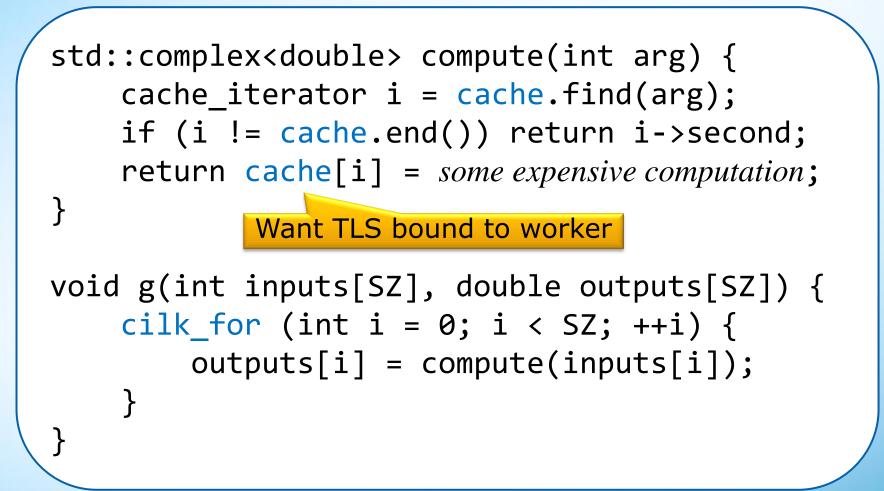
Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



11/2/2012

# **Use case 2: Parallelized with Cilk**





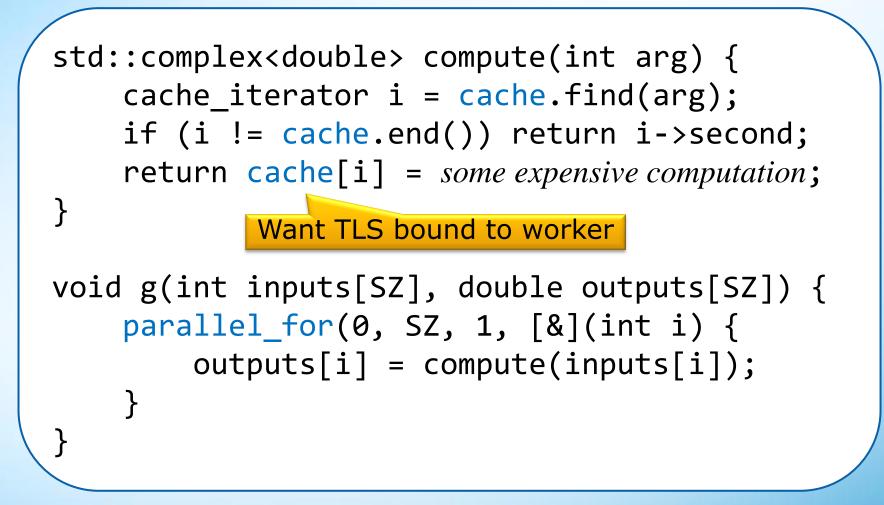
Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



11/2/2012

# **Use case 2: Parallelized with TBB**





Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.

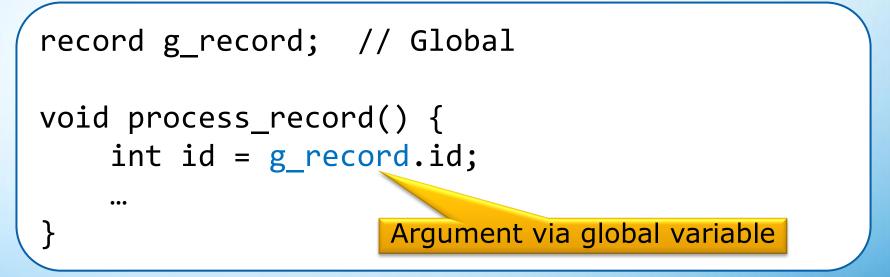


11/2/2012

# **Use case 3: Task-specific variables**

#### The Setup:

We start with a function that receives its argument via a global variable (to avoid parameterproliferation). The value of the global is set in the caller before the call.





Software & Services Group Developer Products Division



# **Use case 3: Serial code**

Process multiple records in a loop.

```
extern record g_record; // Global
void g() {
    for (int i = 0; i < num_recs; ++i) {</pre>
        init record(g_record);
        g record.id = i;
        ...
        process_record(); // Process g_record
```



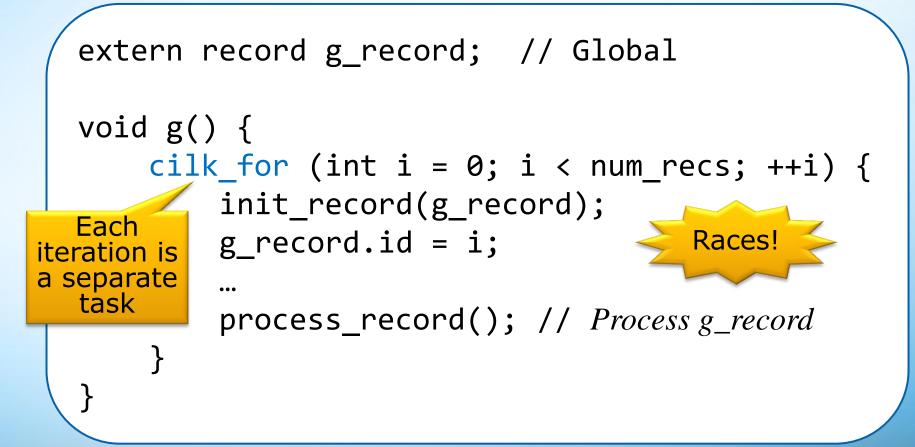
Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



# **Use case 3: Naïve Parallelization**

Process multiple records in a parallel loop.





Software & Services Group Developer Products Division



# **Use case 3: Parallelization with TLS**

Mitigate races using TLS

```
extern thread local record g record;
void g() {
    cilk for (int i = 0; i < num recs; ++i) {</pre>
         init record(g record);
                                        Want per-
         g record.id = i;
                                       worker TLS
                                         (sort of)
         ...
         process_record(); // Process g_record
          Keep last value of g_record. How do we
}
         make other workers destroy their copies?
```



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



# Analysis: Comparing use cases 1 & 2

- Use case 1: Session-specific information
  - All parallel tasks share a common user-level TLS object.
  - The *Thread* in *Thread-local* refers to the user-created std::thread (or main thread), not to the system-created worker thread.
  - If a task writes to the TLS carelessly, it could cause a race.
     (Parallelism can always create races if care is not taken.)
- Use case 2: Dynamic Cache
  - Each worker has its own copy of each TLS object.
  - The *Thread* in *Thread-local* refers to the worker thread, not necessarily the user-created std::thread.
  - Tasks can still race on TLS, but that would require communicating addresses across tasks.



Software & Services Group Developer Products Division



# **Analysis: Use case 3**

- Use case 3: Task-specific variables
  - In the parallel code, each worker has its own copy of each TLS object, as in use case 2.
  - In the serial code, only one object should remain, as in use case 1.
  - Ideally, the T in TLS refers to Task rather than Thread for the specified variable.
  - The Cilk Plus library provides a *holder* hyperobject (similar to a reducer hyperobject) that implements task-local storage.



Software & Services Group Developer Products Division



# What kind of TLS do we need?

- The three use cases described here show that we need:
  - 1. Thread-local storage shared by all the tasks executed by a user-created thread.
  - Worker-local storage that is owned by the system workers and which might survive any given task or usercreated thread.
  - 3. Task-local storage that acts like worker-local storage but is deallocated at the end of a parallel task.



Software & Services Group Developer Products Division



# What should thread\_local specify?

- The thread\_local storage class was introduced in C++11 at the same time as std::thread.
- The concepts of thread and thread-local should be consistent.
- Therefore, thread\_local should specify that the variable is specific to an std::thread (as per use case 1).



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



11/2/2012

# Conclusions

- No single concept of thread local storage suffices for all existing and anticipated use cases.
- We will need a *task* formalism and possibly a *worker* formalism in addition to the existing *thread* formalism.
- User-thread-local, worker-local, and task-local storage should all be available via language and/or library features.



Software & Services Group Developer Products Division





# Software



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



11/2/2012

# **Optimization Notice**

#### **Optimization Notice**

Intel<sup>®</sup> compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel<sup>®</sup> and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel<sup>®</sup> Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel<sup>®</sup> compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel<sup>®</sup> compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel<sup>®</sup> compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel<sup>®</sup> Streaming SIMD Extensions 2 (Intel<sup>®</sup> SSE2), Intel<sup>®</sup> Streaming SIMD Extensions 3 (Intel<sup>®</sup> SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel<sup>®</sup> SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel<sup>®</sup> and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.



Software & Services Group Developer Products Division



# **Legal Disclaimer**

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries. \*Other names and brands may be claimed as the property of others.

Copyright © 2011. Intel Corporation.

#### http://intel.com/software/products



Software & Services Group Developer Products Division

Copyright© 2012, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



11/2/2012