Single Thread Optimization

General Optimization Technique

-O1

- Optimizes for speed
- Disable optimization that increases code size and affect speed

-O2

- Maximize speed (Recommended)
- Auto vectorization is enabled at O2 and higher levels.
- Performs some basic loop optimizations, inlining of intrinsic, Intra-file interprocedural optimization, and most common compiler optimization technologies.

-O3

- Performs O2 optimizations and enables more aggressive loop transformations such as Fusion, Block-Unroll-and-Jam, and collapsing IF statements.
- These optimizations may not cause higher performance unless loop and memory access transformations take place

-Ofast

- Compiler sets options for -O3 -no-prec-div and –fp-model fast=2

-On

- (n>3) Similar to O3

-O0

- No optimization, Used for debugging and correctness.

Optimization Switches

-O1 enables following switches:

- Og: Turns on loops, common sub expression and register optimization
- Os: Optimize source code
- Oy: Omits frame pointer. Suppress creation of frame pointers on cell stack . Frees EBP register for other uses.
- Ob1: Allow functions to be inline
- Gs: Stack probes off. Turn on stack checking (#pragma check_stack)
- Gy: Function level linking. Linker only includes functions referenced in OBJ rather than the entire contents
- Gf: Eliminate duplicate string

-O2, along with above switches it also enables the following:

- Oi: Intrinstic functions. Replace specific functions with inline versions
- Ot: Fast code. Favor optimizations for speed over optimizations for size.

Profile Guided Optimization

Profile Guided Optimization(PGO) **will increase performance** by shrinking code size, reducing branch mispredictions and reduce instruction-cache problems.

Steps to Use PGO

1. Compile code with -prof-gen
   - icc program.c -O2/O3 -o output.out -prof-gen=srcpos -prof-dir=<path>
2. Run Executable File
   - ./output.out
3. Again compile with -prof-use
   - icc program.c -O2/O3 -o output.out -prof-use –ipo -prof-dir=<dir>
   - It will use all .dyn files to generate an optimized executable.

Inter-Procedural Optimization

* Improves performance in program containing many frequently used functions of small and medium length. It is a compiler dependent optimization technique. Can optimize entire program.
* Performs following Operations
  * Eliminate Duplicate Calculations
  * Eliminate In-efficient use of memory simplify loops
  * Dead code elimination
  * Link time Optimization (Compile Program on file-by-file basis)
* Flags Description
  * -ipo To optimize whole program
  * -ip To optimize single file
* Note: Increases compile time.

Floating Point Optimization

Objective of floating point application may vary from

1. Accuracy
   * Produce results that are "close" to the result of the exact calculation
   * Usually measured in fractional error, or sometimes "units in the last place".
2. Reproducibility
   * Produce consistent results:
   * From one run to the next;
   * From one set of build options to another;
   * From one compiler to another
   * From one processor or operating system to another
3. Performance
   * Produce an application that runs as fast as possible

Floating Point Optimization Options

- *-fp-model*
  - =precise — To get value-safe optimization (Similar to source)
  - =except — Strict floating point optimization
  - =strict — Code reassociation optimizations
  - =fast=1 (default) — Unsafe optimizations
  - =fast=2 — Some additionl optimizations are allowed and enables -fimf-domainexclusion switch if its xeon phi.
    =(-O2 -ipo -xHost -no-prec-div)
- *-[no-]fast-transcendentals*
  - Enable[Disable] use of fast math functions
- *-fimf-precision*
  - Safe optimization of math functions
- *-fimf-arch-consistency=true*
  - If reproducibility between different processors of same architecture is important.
- *-no-prec-div*
  - Consistent, correctly rounded results for division
- *-no-prec-sqrt*
  - Consistent, correctly rounded results for square root.

Floating Point Optimization

* -fp-model=precise disables the following optimizations which is used in other -fp-model which favors speed over accuracy.
  * reassociation e.g(a + b) + c = a + (b + c)
  * zero folding e.g.X+0= X, X*0 = 0
  * multiply by reciprocal e.g. A/B = A*(1/B) (Division will take extra time compared to multiplication operation)
  * approximate square root
  * abrupt underflow (flush-to-zero)
  * drop precision of RHS to that of LHS and etc
* -fp-model = [precise|source] improves the consistency and reproducibility of floating point results while limiting the impact on performance.
* Floating point contraction (Fused Multiply-Add,FMA) is enabled in all generalized optimization >-O1 and also Intel Xeon processor below V2 doesnot have FMA.
* -fp-model-strict or -no-fma disables FMA

Math Kernel Library(MKL)

Math Kernel Library (MKL) is library of optimized math routines for science, engineering, and financial applications.

* Supports 32-bit, 64-bit Intel® or compatible processor and Intel® MIC.
* Fortran and C/C++ are broadly supported language.
* Automatically offloads if Intel® Xeon Phi™ is available.
* Optimized for Intel® Architecture.
* MKL uses OpenMP for threading.

Intel® MKL Contents

BLAS	Basic Linear Algebra subroutines
	Vector-vector, matrix vector and matrix operations
LAPACK	Linear Algebra Package
	Solvers and Eigen solvers.
DFTs	Discrete Fourier transforms
	Mixed radix, multi-dimensional transforms
	Multithreaded
VML	Vector Math Library
	Set of vectorized transcendental functions
	Most of libm functions, but faster
VSL	Vector Statistical Library
	Set of vectorized random number generators

MKL Example

* To multiply in_A and in_B where out_C will be resultant:

```
//Sequentially
void mat_multiply(double **in_A, double  **in_B, double  **out_B, int
ROWA, int COLA, int COLB)
{ for (row=0; row<ROWA; row++)
                for (column=0; column<COLB; column++)
                        for (column1=0; column1<COLA; column1++)
                                out_C[row*ROWA+column] +=
in_A[row*ROWA+column1] * in_B[column1*COLA+column];
}
```

```
//Using MKL
void mat_multiply(double **in_A, double  **in_B, double  **out_B, int
ROWA, int COLA, int COLB)
{cblas_dgemm(CblasRowMajor, CblasRowMajor,COLB, COLA, 1.0, in_A,
COLB, in_B, COLA, 0.0, out_C, ROWA);}
```

MKL Example (cont..)

```
[test@phi Training]$ export MKL_MIC_ENABLE=1
[test@phi Training]$ export OFFLOAD_REPORT=2
[test@phi Training]$ icc mm_mkl.c -mkl -o mm_mkl.out
[test@phi Training]$ ./mm_mkl.out
[MKL] [MIC --] [AO Function]    DGEMM
[MKL] [MIC --] [AO DGEMM Workdivision]  0.08 0.92
[MKL] [MIC 00] [AO DGEMM CPU Time]      11.708980 seconds
[MKL] [MIC 00] [AO DGEMM MIC Time]      9.261564 seconds
[MKL] [MIC 00] [AO DGEMM CPU->MIC Data] 1626435968 bytes
[MKL] [MIC 00] [AO DGEMM MIC->CPU Data] 3173852160 bytes
Multiplied using Intel(R) MKL
```

Intel® Threading Building Blocks(TBB)

The library consists of data structures and algorithms that allow a programmer to avoid some complications arising from the use of native threading packages in which individual threads of execution are created, synchronized, and terminated manually.

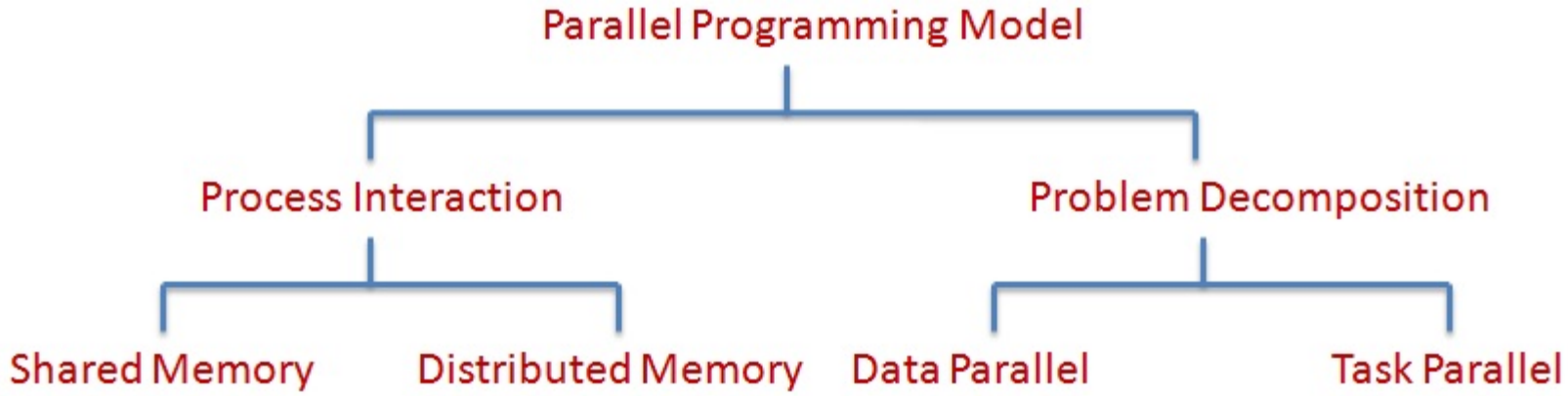
* Supports 32 and 64 bit architecture and also android OS.
* Supports C++
* Higher level task based parallelism
* Also available as open source

Intel® Threading Building Blocks(TBB) (contd..)

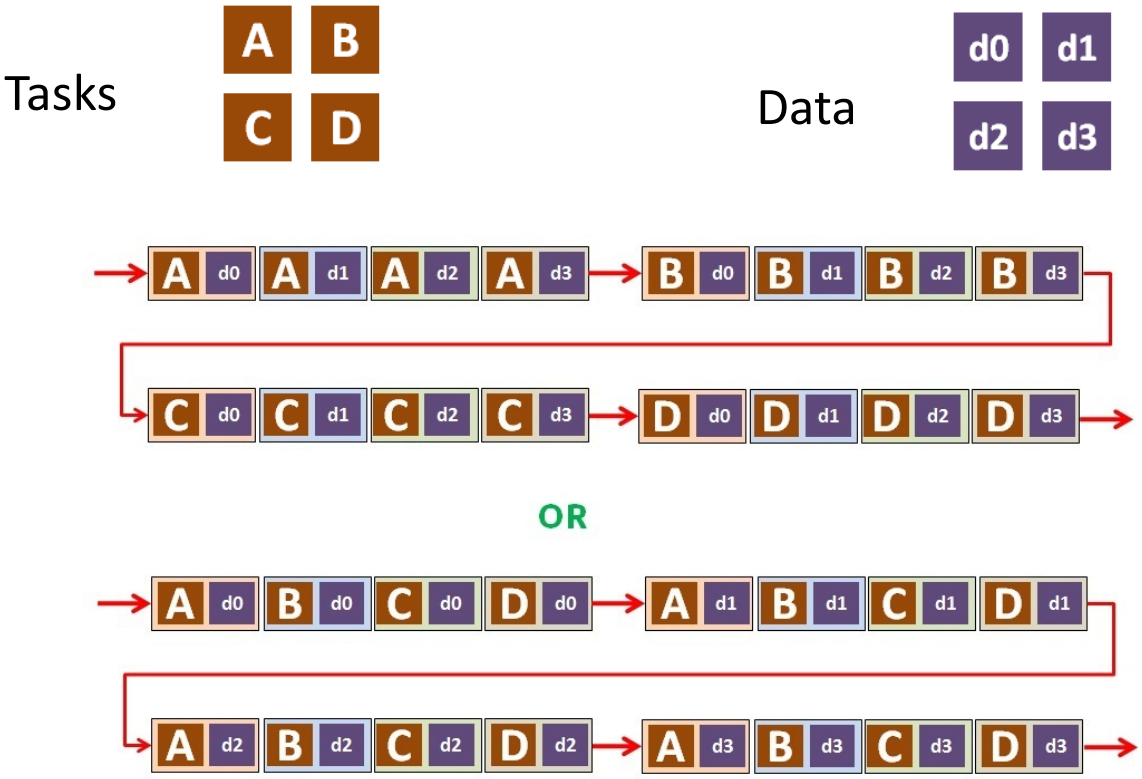
* Dynamic memory functions are replaced by TBB library functions by loading proxy library at run-time or by linking with the proxy library.
  * These functions include:
    * C library: malloc, calloc, realloc, free
    * Standard POSIX* function: posix_memalign
    * Obsolete functions: valloc, memalign, pvalloc, mallopt
    * Global C++ operators new and delete.
* To use TBB library functions link *-ltbb* option while compiling.
* Scalable allocator (tbbmalloc) of TBB allocates and frees memory in a way that scales with the number of processors. (*-ltbb –ltbbmalloc* are the libraries to link)

Parallel Programming Models

* Parallel Programming model refers to *model* for writing parallel computer applications which can be compiled and run, in consideration of the underlying hardware system and inherent sequential nature of the programming language

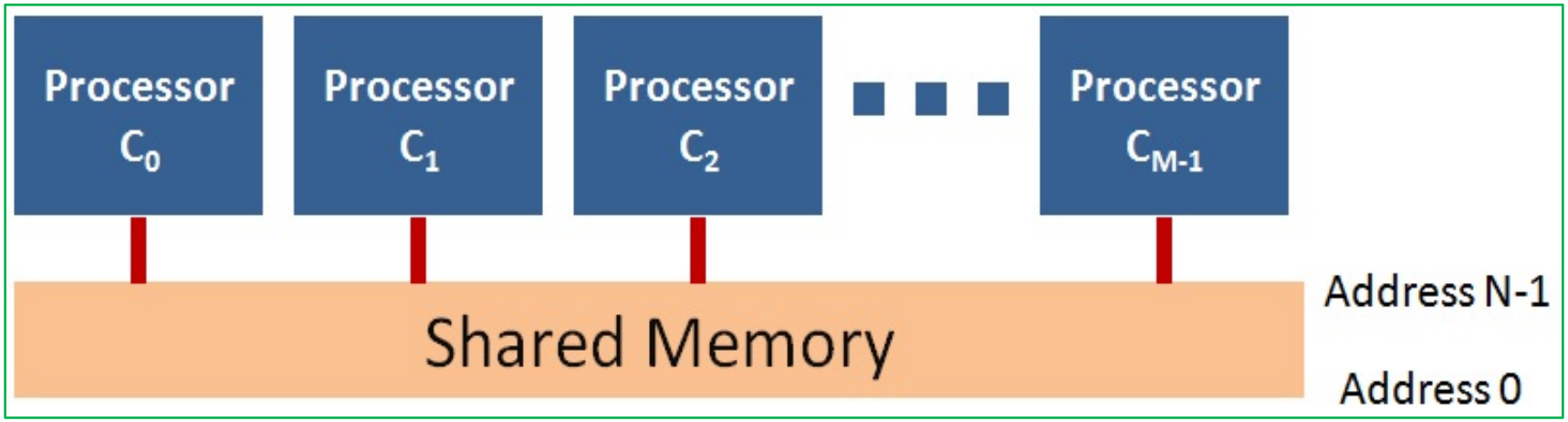


Tasks/Data Sequential Program



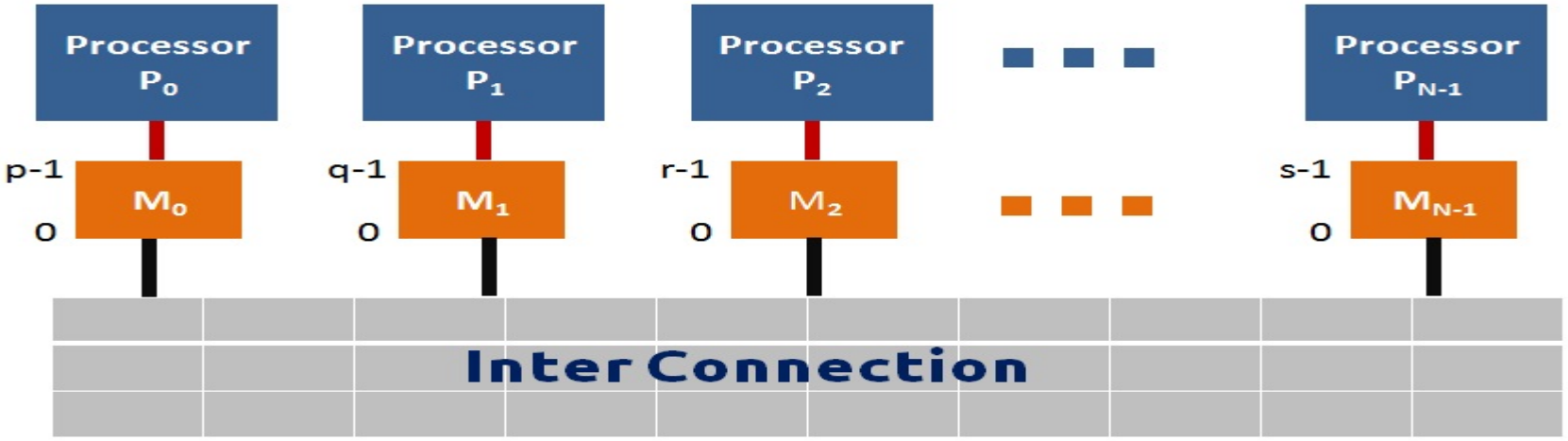
Shared Memory Model

* Processors share a single memory (Single Memory Address Space)
* Also referred to as Multi-Processor system
* A single copy of operating system controls all the processors
* All processors have access to all parts of memory, may or may-not in parallel
* 2 Parallel Programming Approaches OpenMP and MPI



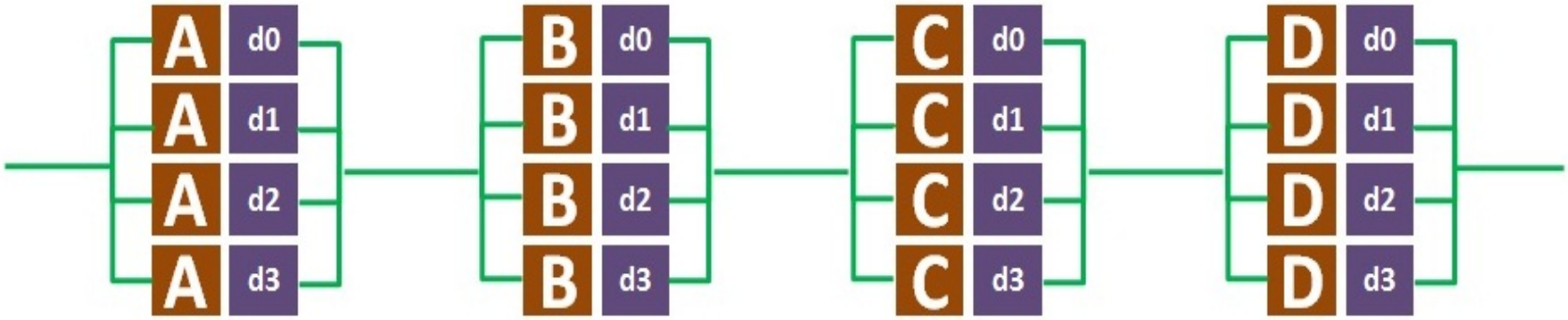
Distributed Memory Model

* Processors have exclusive individual memory
* Also referred to as Many-Processor system
* Each processor system is controlled by its own copy of operating system
* Processors with their own memory are connected together with an interconnect fabric (Ethernet, Infiniband, etc.)
* 1 Parallel Programming approach MPI



Data parallel Model

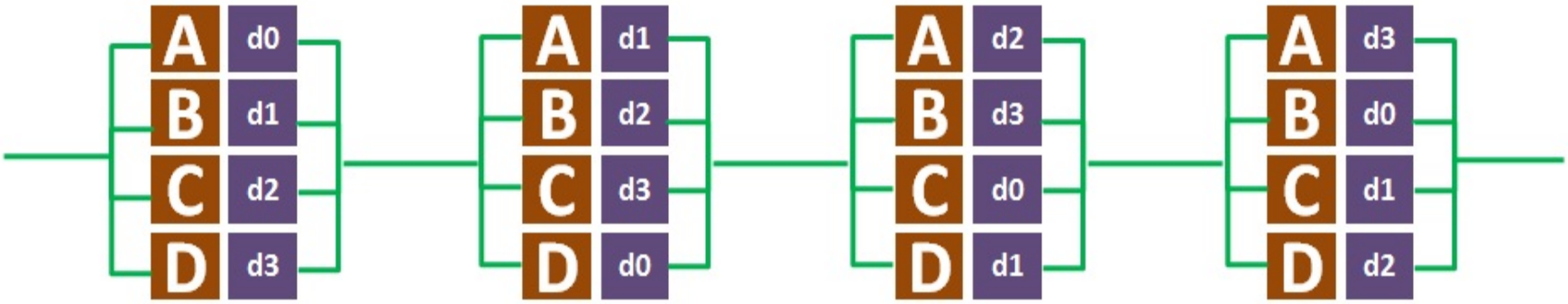
* An *operation* or *set of operations* performed independently on a data-set
* In Shared Memory Systems, the data-set may be accessible to all operating processors, but restricted to separate portions (Mostly structured array)
* In Distributed Memory Systems, the data-set is divided among memories and operated on locally



Data Parallel

Task Parallel Model

* Multiple distinct *operation* performed independently on data-set
* Need arises to be able to communicate between different operating processes



Task Parallel

Questions

BACK-UP

Profile Guided Optimization (contd)

Programmer needs to make sure that program can exit in order to generate profiler.

In the embedded system, if the program is running infinitely without an exit point, some addition work needs to be done to make sure the profiler generation.

1. Manually adding exit code in the program.
2. Adding the PGO API _PGOPTI_Prof_Dump_All() in the source to dump the profiler.
3. Using environment to make regular dumps, for example dump all in one file every 5000 microsecond by using following environment:
   * export INTEL_PROF_DUMP_INTERVAL 5000
   * export INTEL_PROF_DUMP_CUMULATIVE 1