Intel Xeon Phi

Israel Silva Barbará

Professor Adenauer Yamin

- Indice

- Princípio Geral de Funcionamento
- Exemplos de Aplicações
- Modelo de Programação
- Linguagens Suportadas
- Exemplos de Produtos no Mercado

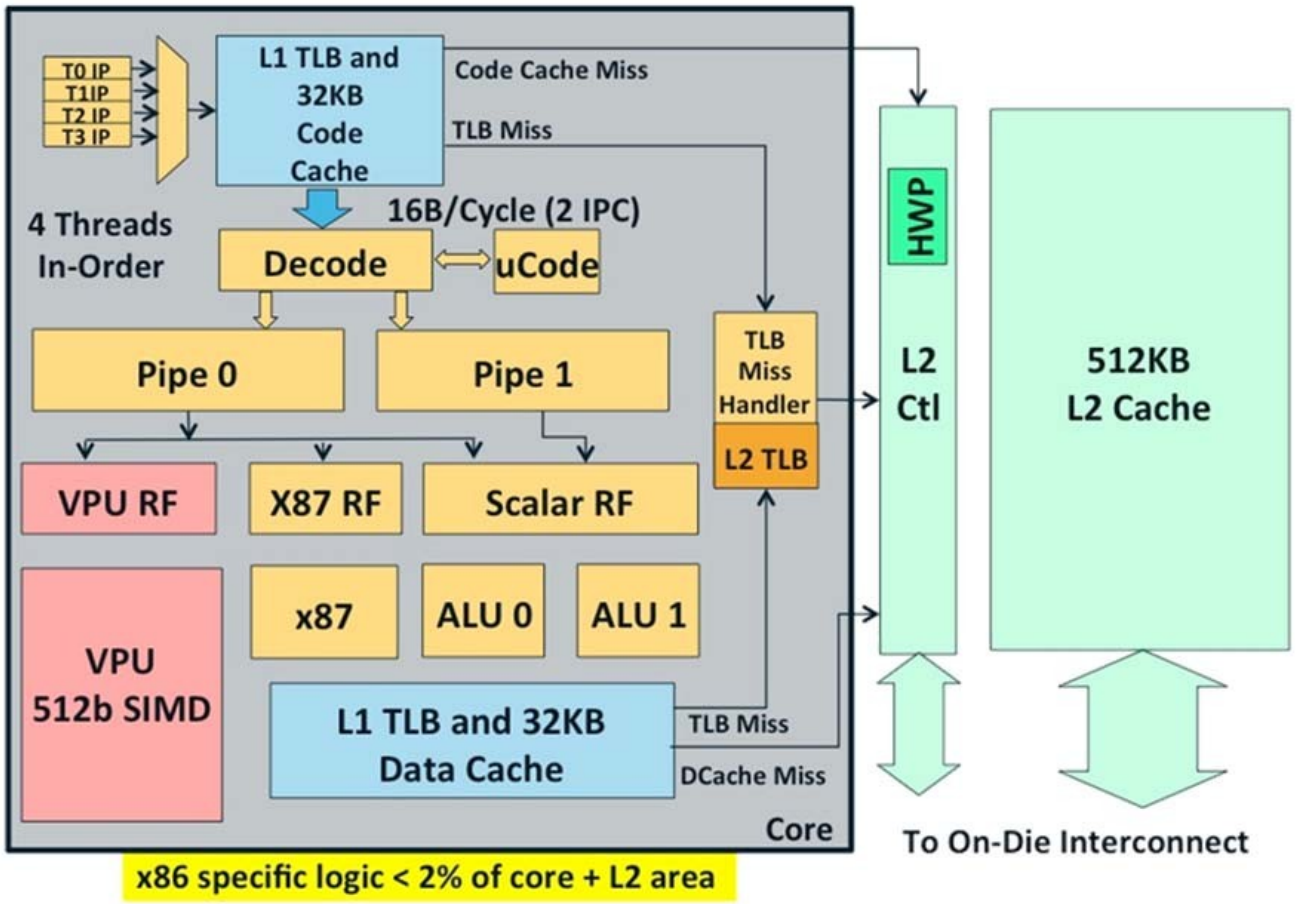
Princípio Geral de Funcionamento

- CoProcessador
- Roda Sistema Linux integrado
- Arquitetura X86 – com suporte para 64bit instruções
- Mais de 50 Nucleos , Hipertreading 4x1
- Mais de 200 Threads
- Compatível com processadores Xeon
- Usa vetor de 512bits (8 * 64) = oito computações de ponto flutuante de dupla precisão por clock.

Princípio Geral de Funcionamento

- Programas precisam somente ser recompilados para rodar no coprocessador
- Modo de compilaçao:
  - Nativo : Programa executa tanto no processador xeon como no coprocessador Phi
  - Off-Load : Programa executa no processador , mandando tarefas para o co-processador

Princípio Geral de Funcionamento



•Exemplos de Aplicações

- simulação de MonteCarlo
- Equação diferencial Black-Scholes
- HPL
- LifeSc
- reverse time migration
- Ray-tracing,
- Stream

Linguagens Suportadas

- C
- C++
- Fortran

Modelo de Programação

- Existem algumas recomendações que podem ser feitas para desenvolvedores.
-
- Para programadores Fortran , use OpenMP, DO CONCORRENTE e MPI.
- Para programadores C++ , utilize Intel TBB, Intel Cilk Plus e OpenMP.
- Para programadores C, utilize OpenMP e Intel Cilk Plus.

Modelo de Programação

Summing vector elements in C using OpenMP – openmp.org

```
#pragma omp parallel for reduction(+: s)
for (int i = 0; i < n; i++) {
  s += x[i];
}
```

Per element multiply in C++ using Intel® Array Building Blocks – intel.com/go/arbb

```
void products( const arbb::dense<arbb::f32>& a,
               const arbb::dense<arbb::f32>& b,
               arbb::dense<arbb::f32>& c) {
  c = a * b;
}
```

Dot product in Fortran using OpenMP – openmp.org

```
!$omp parallel do reduction ( + : adotb )
  do j = 1, n
    adotb = adotb + a(j) * b(j)
  end do
!$omp end parallel do
```

MPI code in C for clusters – intel.com/go/mpi

```
for (d=1; d<ntasks; d++) {
  rows = (d <= extra) ? avrow+1 : avrow;
  printf(" sending %d rows to task %d\n", rows, dest);
  MPI_Send(&offset, 1, MPI_INT, d, mtype, MPI_COMM_WORLD);
  MPI_Send(&rows, 1, MPI_INT, d, mtype, MPI_COMM_WORLD);
  MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, d, mtype, MPI_COMM_WORLD);
  MPI_Send(&b, NCA*NCB, MPI_DOUBLE, d, mtype, MPI_COMM_WORLD);
  offset = offset + rows;
}
```

Matrix Multiply in Fortran using Intel® Math Kernel Library – intel.com/software/products

```
call DGEMM(transa,transb,m,n,k,alpha,a,lda,b,ldb,beta,c,ldc)
```

Sum in Fortran, using co-array feature – intel.com/software/products

```
REAL SUM[*]
CALL SYNC_ALL( WAIT=1 )
DO IMG= 2,NUM_IMAGES()
  IF (IMG==THIS_IMAGE()) THEN
    SUM = SUM + SUM[IMG-1]
  ENDIF
  CALL SYNC_ALL( WAIT=IMG )
ENDDO
```

Parallel function invocation in C using Intel® Cilk™ Plus – cilk.org

```
cilk_for (int i=0; i<n; ++i) {
  Foo(a[i]);
}
```

Parallel function invocation in C++ using Intel® Threading Building Blocks – threadingbuildingblocks.org

```
parallel_for (0, n,
  [=](int i) { Foo(a[i]); }
);
```

Per element multiply in C using OpenCL – intel.com/go/opencl

```
kernel void
  dotprod( global const float *a,
           global const float *b,
           global float *c) {
    int myid = get_global_id(0);
    c[myid] = a[myid] * b[myid];
}
```

Bibliografia

- http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
- http://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors.pdf
- http://streamcomputing.eu/blog/2012-11-12/intels-answer-to-amd-and-nvidia-the-xeon-phi-5110p/
- http://www.slideshare.net/andres_gomez_tato/do-pro-hpcast2013v2