**PROFILE, OPTIMIZE, PROFILE, OPTIMIZE, RINSE AND REPEAT**

```bash
jsrun -n $nmpl -g 1 nvprof -o clover.bm32.%p.prof  <path>/clover_leaf

nvprof -i clover.bm32.33405.prof >& clover.bm32.33405.txt

$ cat clover.bm32.33405.txt
======== Profiling result: 

<table>
<thead>
<tr>
<th>Type</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities</td>
<td>7.54%</td>
<td>13.5475s</td>
<td>2955</td>
<td>4.5846ms</td>
<td>4.5446ms</td>
<td>4.6170ms</td>
</tr>
<tr>
<td></td>
<td>6.02%</td>
<td>10.8087s</td>
<td>2955</td>
<td>3.6578ms</td>
<td>3.6069ms</td>
<td>3.6777ms</td>
</tr>
<tr>
<td></td>
<td>5.75%</td>
<td>10.3340s</td>
<td>2955</td>
<td>3.4971ms</td>
<td>3.4545ms</td>
<td>3.5179ms</td>
</tr>
<tr>
<td></td>
<td>4.88%</td>
<td>8.76880s</td>
<td>5910</td>
<td>1.4837ms</td>
<td>1.4514ms</td>
<td>1.5110ms</td>
</tr>
<tr>
<td></td>
<td>4.80%</td>
<td>8.62393s</td>
<td>5910</td>
<td>1.4592ms</td>
<td>1.4286ms</td>
<td>1.4850ms</td>
</tr>
<tr>
<td></td>
<td>4.74%</td>
<td>8.50618s</td>
<td>6208</td>
<td>1.3702ms</td>
<td>1.3358ms</td>
<td>1.4000ms</td>
</tr>
<tr>
<td></td>
<td>4.44%</td>
<td>7.98215s</td>
<td>5910</td>
<td>1.3506ms</td>
<td>1.3307ms</td>
<td>1.3687ms</td>
</tr>
<tr>
<td></td>
<td>4.34%</td>
<td>7.79325s</td>
<td>5910</td>
<td>1.3187ms</td>
<td>1.2977ms</td>
<td>1.3403ms</td>
</tr>
<tr>
<td></td>
<td>4.24%</td>
<td>7.61488s</td>
<td>2955</td>
<td>2.5769ms</td>
<td>2.5658ms</td>
<td>2.5897ms</td>
</tr>
<tr>
<td></td>
<td>4.23%</td>
<td>7.59213s</td>
<td>2955</td>
<td>2.5692ms</td>
<td>2.5471ms</td>
<td>2.5891ms</td>
</tr>
<tr>
<td></td>
<td>3.91%</td>
<td>7.02675s</td>
<td>2955</td>
<td>2.3779ms</td>
<td>2.3744ms</td>
<td>2.3961ms</td>
</tr>
<tr>
<td></td>
<td>3.91%</td>
<td>7.02292s</td>
<td>2955</td>
<td>2.3766ms</td>
<td>2.3632ms</td>
<td>2.3954ms</td>
</tr>
</tbody>
</table>
```

............
OPENACC DIRECTIVES FOR GPUS

% pgfortran -fast -ta=tesla -Minfo -c PdV_kernel.f90

dv_kernel:

... 77, Loop is parallelizable
79, Loop is parallelizable

Accelerator kernel generated
Generating Tesla code

77, !$acc loop gang, vector(4) ! blockid%y
! threadid%y
79, !$acc loop gang, vector(32) ! blockid%x
! threadid%x

...
LOOP SCHEDULING

Collapse:

 !$ACC KERNELS LOOP INDEPENDENT COLLAPSE(2)
     DO k=y_min,y_max
         DO j=x_min,x_max
             left_flux = (xarea(j,k)*(xvel0(j,k)+xvel0(j,k+1)+xvel0(j,k)+xvel0(j,k+1)))*0.25_8*dt*0.5
             ....
         ENDDO
     ENDDO

Explicit loop schedules:

 !$ACC KERNELS LOOP INDEPENDENT GANG
     DO k=y_min,y_max
 !$ACC LOOP INDEPENDENT VECTOR(512)
     DO j=x_min,x_max
         left_flux = (xarea(j,k)*(xvel0(j,k)+xvel0(j,k+1)+xvel0(j,k)+xvel0(j,k+1)))*0.25_8*dt*0.5
         ....
     ENDDO
     ENDDO
MEMORY COALESCING

**Coalesced access:**

A group of 32 contiguous threads ("warp") accessing adjacent words

Few transactions and high utilization

**Uncoalesced access:**

A warp of 32 threads accessing scattered words

Many transactions and low utilization

For best performance `threadIdx.x` should access contiguously
async(n): launches work asynchronously in queue \( n \)

wait(n): blocks host until all operations in queue \( n \) have completed

Can significantly reduce launch latency, enables pipelining and concurrent operations

```plaintext
#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)
  ...

#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)
  ...

#pragma acc wait(1)
```
PINNED MEMORY

% pgfortran -fast -ta=tesla:pinned -Minfo -c PdV_kernel.f90

- Data transfers between the host and device must be done from CPU non-paged ("pinned") memory. By default, PGI creates buffers in pinned memory to perform the transfer.
- Using "pinned" the host data is allocated directly in non-paged memory eliminating the need to copy data from virtual to pinned memory.
- Caveats:
  - Need enough physical memory for the pinned memory
  - Allocation of pinned memory has a high overhead cost
- Best used when data is transferred often but has few allocations/deallocations
Programming GPU-Accelerated Systems
CUDA Unified Memory for Dynamically Allocated Data
PGI OpenACC and CUDA Unified Memory

Compiling with the -ta=tesla:managed option

```c
#pragma acc data copyin(a,b) copyout(c)
{
    ...
    #pragma acc parallel
    {
        #pragma acc loop gang vector
        for (i = 0; i < n; ++i) {
            c[i] = a[i] + b[i];
            ...
        }
    }
    ...
}
```

C malloc, C++ new, Fortran allocate all mapped to CUDA Unified Memory

GPU Developer View With
CUDA Unified Memory
PGI OpenACC and CUDA Unified Memory

Compiling with the `-ta=tesla:managed` option

...#pragma acc parallel
{
#pragma acc loop gang vector
    for (i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
        ...
    }
}
...

C `malloc`, C++ `new`, Fortran `allocate` all mapped to CUDA Unified Memory
GTC Performance using OpenACC

OpenPOWER | NVLink | Unified Memory | P100 | V100

- IBM POWER8NVL, 2 sockets, 20 cores, NVLINK
- No Data Directives in sources, compiled with -ta=tesla:managed
## PGI OPENACC POOL ALLOCATOR

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGI_ACC_POOL_ALLOC</td>
<td>Disable the pool allocator. The pool allocator is enabled by default; to disable it, set PGI_ACC_POOL_ALLOC to 0.</td>
</tr>
<tr>
<td>PGI_ACC_POOL_SIZE</td>
<td>The default size is 1GB but other sizes can be used. The actual pool size is set such that the size is the nearest, smaller number in the Fibonacci series compared to the provided or default size. If necessary, the pool allocator will add more pools but only up to the PGI_ACC_POOL_THRESHOLD value.</td>
</tr>
<tr>
<td>PGI_ACC_POOL_THRESHOLD</td>
<td>Set the percentage of total device memory that the pool allocator can occupy. The default is set to 50% but other percentages can be used.</td>
</tr>
<tr>
<td>PGI_ACC_POOL_ALLOC_MAXSIZE</td>
<td>Set the maximum size for allocations. The default maximum size for allocations is 500MB but another size (i.e., 100KB, 10MB, 250MB, etc.) can be used as long as it is greater than or equal to 16B.</td>
</tr>
<tr>
<td>PGI_ACC_POOL_ALLOC_MINSIZE</td>
<td>Set the percentage of total device memory that the pool allocator can occupy. The default is set to 50% but other percentages can be used.</td>
</tr>
</tbody>
</table>
DO CONCURRENT (k=y_min:y_max, j=x_min:x_max) &
    LOCAL (right_flux,left_flux,top_flux,bottom_flux,total_flux, &
        min_cell_volume,energy_change,recip_volume)

left_flux  =  (xarea(j,k )*(xvel0(j,k )+xvel0(j,k+1) &
    +xvel0(j,k )+xvel0(j,k+1))*(0.25_8*0.5 &
right_flux = (xarea(j+1,k )*xvel0(j+1,k+1) &
    +xvel0(j+1,k+1)+xvel0(j+1,k ))*(0.25_8*0.5 &
bottom_flux=yarea(j,k )*yvel0(j,k )+yvel0(j,k+1) &
    +yvel0(j,k+1)+yvel0(j,k+1))*(0.25_8*0.5 &
top_flux  =  (yarea(j ,k+1)*yvel0(j,k +yvel0(j+1,k+1) &
    +yvel0(j,k+1)+yvel0(j+1,k+1))*(0.25_8*0.5 &
total_flux=right_flux-left_flux+top_flux-bottom_flux}&
volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)

min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
    ,volume(j,k)+right_flux-left_flux &
    ,volume(j,k)+top_flux-bottom_flux)
recip_volume=1.0/volume(j,k)
energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...energy1(j,k)=energy0(j,k)-energy_change
density1(j,k)=density0(j,k)*volume_change(j,k)

ENDO