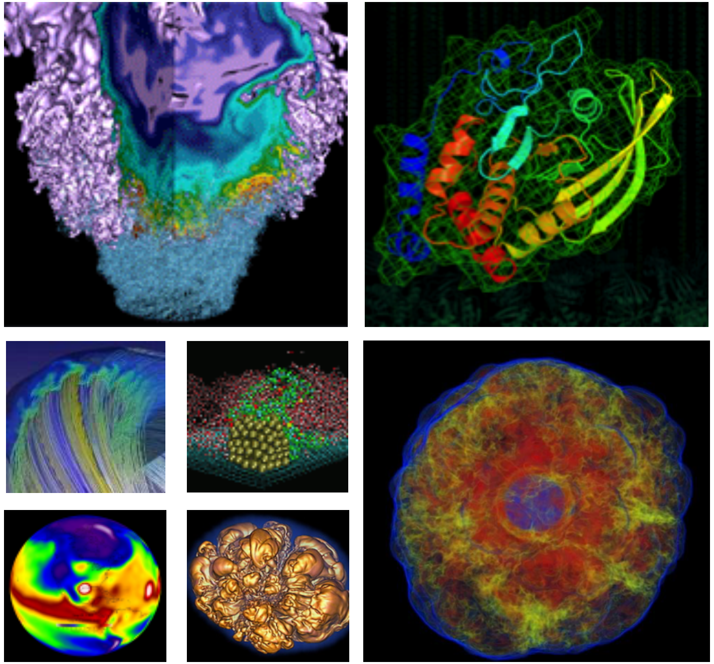
Comparing Managed Memory and ATS on Volta GPUs



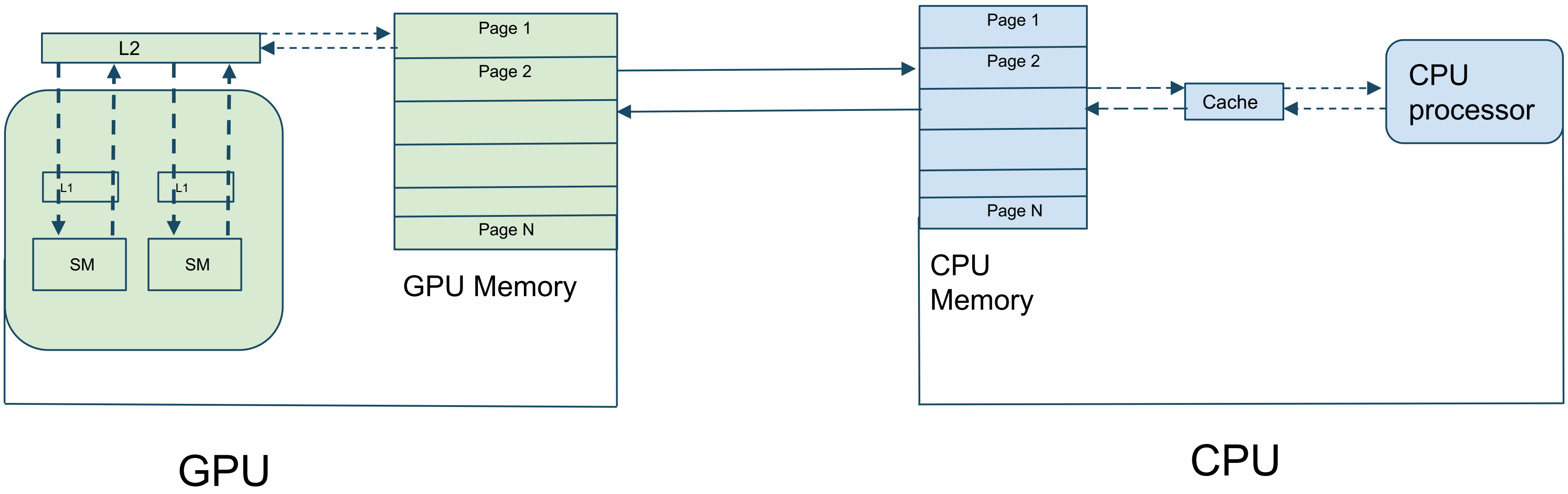
NERSC

Rahul Gayatri, Kevin Gott, Jack Deslippe @ SC 2019 (PMBS19)

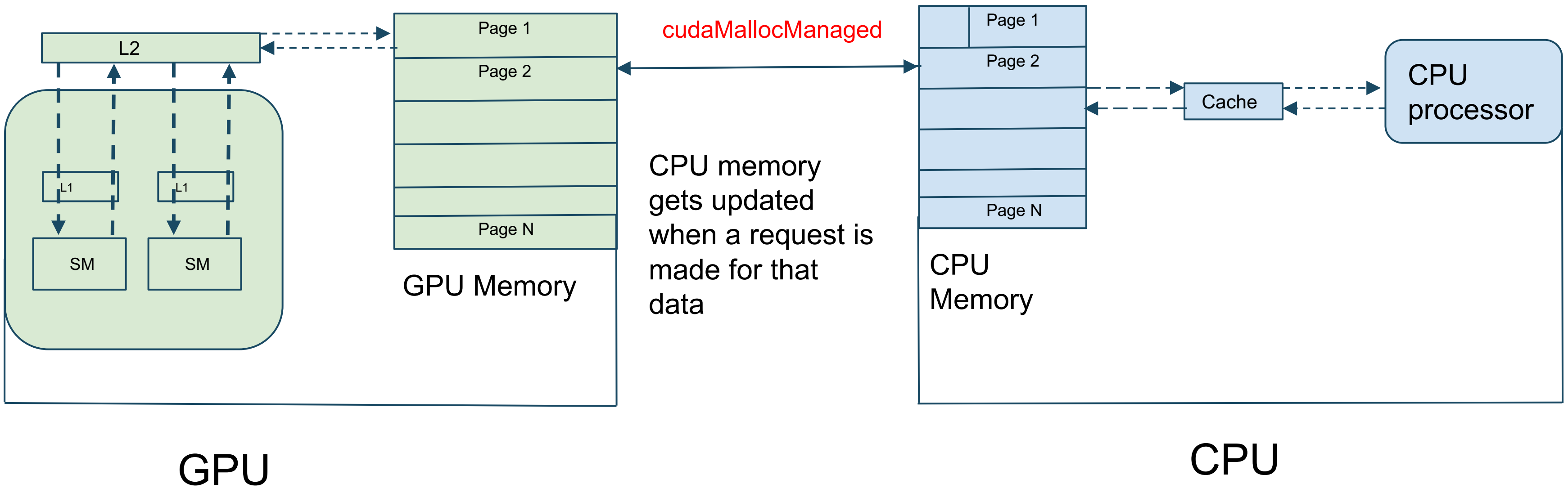
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

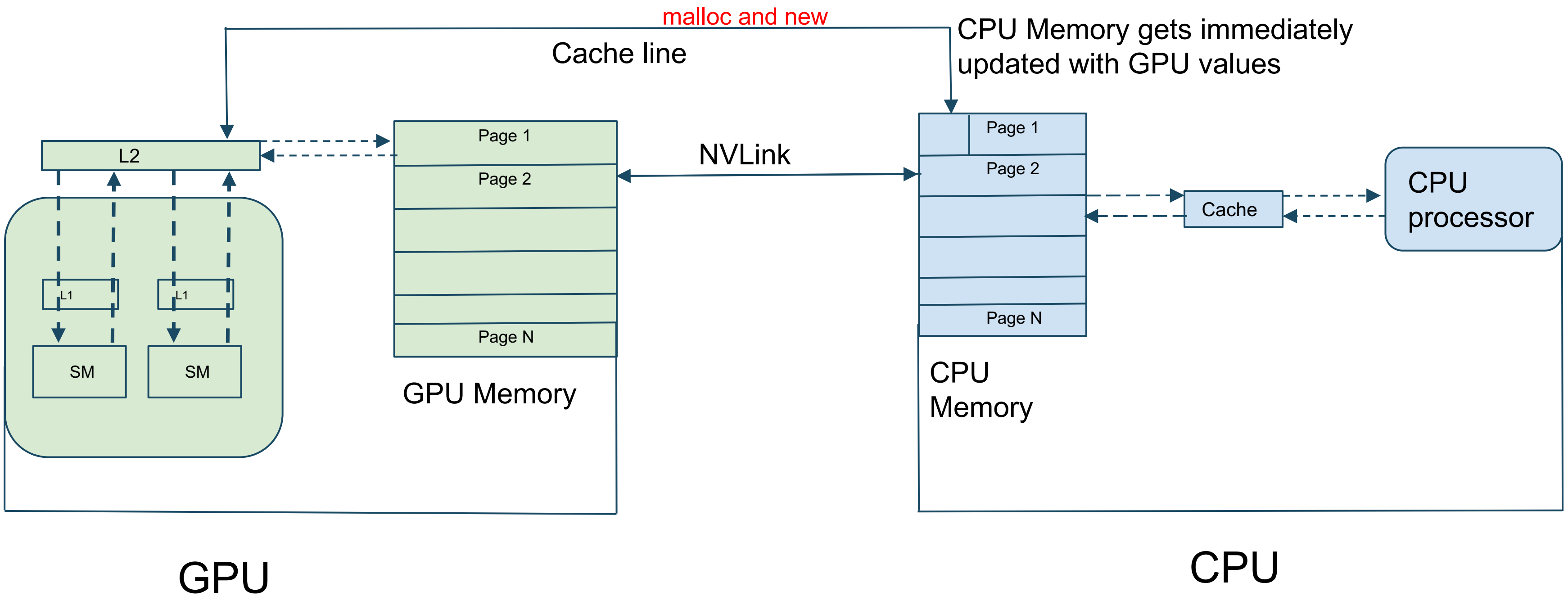
CPU and GPU architecture



Managed implementation



ATS implementation



ATS available on V100 + P9 connected via NVLink

Managed vs ATS

Managed

- cudaMallocManaged
- Granularity of data transfer - page size
- Data back on CPU when needed
- Available since cuda/6.0

Address Translation Service (ATS)

- malloc and new
- Granularity of data transfer - cache line
- Cache coherency on CPU
- GPU accesses entire CPU page tables
- Available since cuda/9.2

Experimental Pseudo code

```
//x[N][M], y[N][M]
#if managed_memory
cudaMallocManaged(&x,N*M*sizeof(double));
cudaMallocManaged(&x,N*M*sizeof(double));
#elif defined(ATS)
x = (double*) malloc(N*M*sizeof(double));
y = (double*) malloc(N*M*sizeof(double));
#endif
```

```
for(outer)//GPU-CPU toggle
{
  for(inner)//consecutive GPU kernel launches
  {
     //N = 80 (number of SMs in V100)
    DAXPY<<<N,32>>>(x,y);
  } //end inner
  TouchOnCPU(y);
}//end outer
```

Experimental Parameters

```
for(outer)//GPU-CPU toggle
{
  for(inner)//consecutive GPU kernel launches
  {
     //N = 80 (number of SMs in V100)
    DAXPY<<<N,32>>>(x,y);
  } //end inner
  TouchOnCPU(y);
}//end outer
```

- Continuous transfer of data between CPU-and-GPU
- Effects of continuous GPU memory accesses
- Size of data

Metrics studied

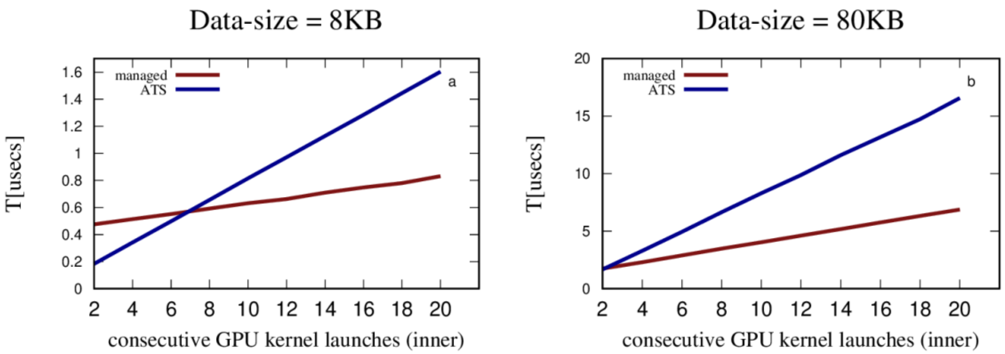
```
for(outer)//GPU-CPU toggle
{
  for(inner)//consecutive GPU kernel launches
  {
     //N = 80 (number of SMs in V100)
    DAXPY<<<N,32>>>(x,y);
  } //end inner
  TouchOnCPU(y);
}//end outer
```

- Continuous transfer of data between CPU-and-GPU
- Effects of continuous GPU memory accesses
- Size of data

- **DAXPY performance**
- **TouchOnCPU performance**
- **Prefetch vs non-prefetch**
- **Total performance**

Managed is better for more GPU work

CPU-GPU toggles (outer) = 2

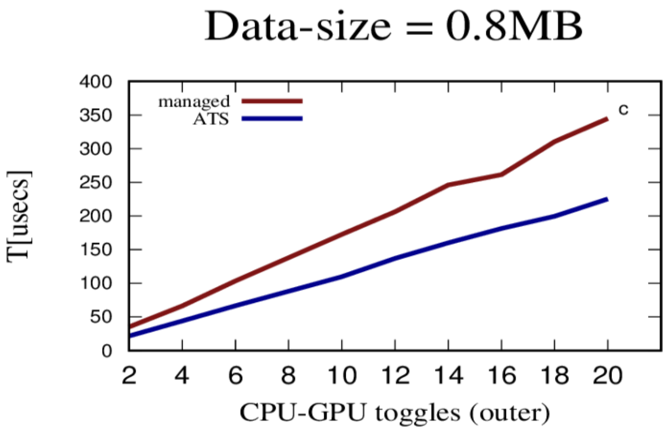


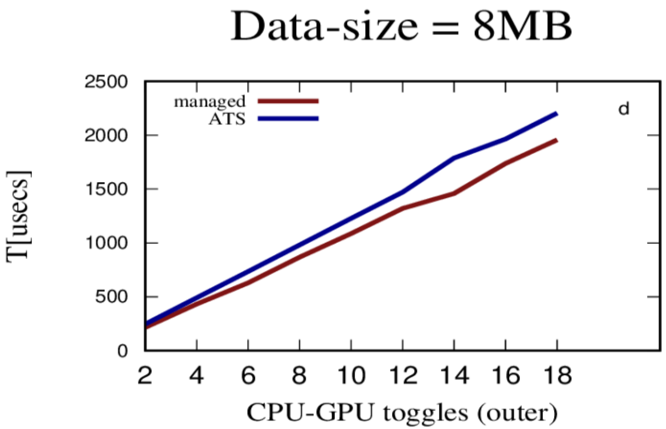
- ATS better for low number of consecutive GPU kernel launches and **small data sizes**
- Managed memory has a **higher initial cost**
- Managed **slope is lower** than ATS
- **Data always on GPU** for managed after the first kernel launch

Data size - data processed by each threadblock

Managed better with higher data sizes

Consecutive GPU kernel launches (inner) = 2

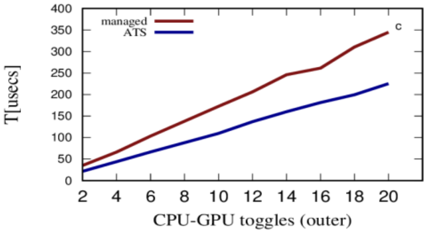




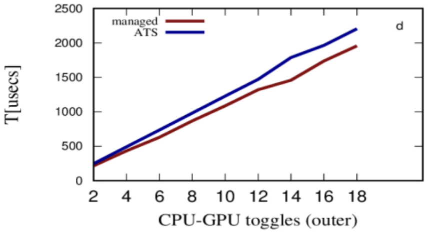
Consecutive GPU accesses more important

Consecutive GPU kernel launches (inner) = 2

Data-size = 0.8MB

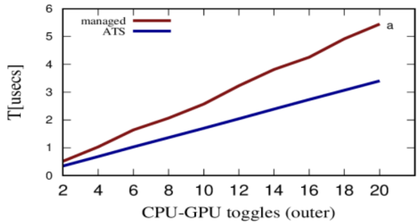


Data-size = 8MB

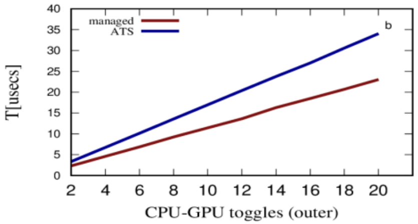


Consecutive GPU kernel launches (inner) = 4

Data-size = 8KB



Data-size = 80KB



- As data size increases managed memory is faster than ATS
- For smaller data sizes with fewer number of consecutive GPU accesses ATS is better than managed
- Number of consecutive GPU accesses is more important than frequency of CPU accesses

Prefetch directive

- cudaMemPrefetchAsync(void* devPtr, size_t size, int dstDevice, cudaStream_t stream)
  - dstDevice - GPU number
  - cudaCpuDeviceId - CPU

```
cudaMemPrefetchAsync(x, N*M*sizeof(double),gpuDeviceId,0);
for(outer)
{
  cudaMemPrefetchAsync(y, N*M*sizeof(double),gpuDeviceId,0);
  for(inner)
  {
    DAXPY<<<N,32>>>(x,y);
  } //end inner
  cudaMemPrefetchAsync(y, N*M*sizeof(double),cudaCPUDeviceId,0);
  TouchOnCPU(y);
}//end outer
```

Managed vs ATS (T[microsecs])

CPU-GPU toggles (outer) = 10 : Consecutive GPU kernel launches (inner) = 2

Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	172.5	65.4 (2.6x)	109.7	73.8 (1.4x)

CPU-GPU toggles (inner) = 2 : Consecutive GPU kernel launches (inner) = 10

Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	55.3	34.5 (1.6x)	126.5	37.13 (3.4x)

Managed+prefetch vs ATS+prefetch

CPU-GPU toggles (outer) = 10 : Consecutive GPU kernel launches (inner) = 2

Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	172.5	65.4 (2.6x)	109.7	73.8 (1.4x)

CPU-GPU toggles (inner) = 2 : Consecutive GPU kernel launches (inner) = 10

Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	55.3	34.5 (1.6x)	126.5	37.13 (3.4x)

T[Managed+prefetch] < T[ATS+prefetch]

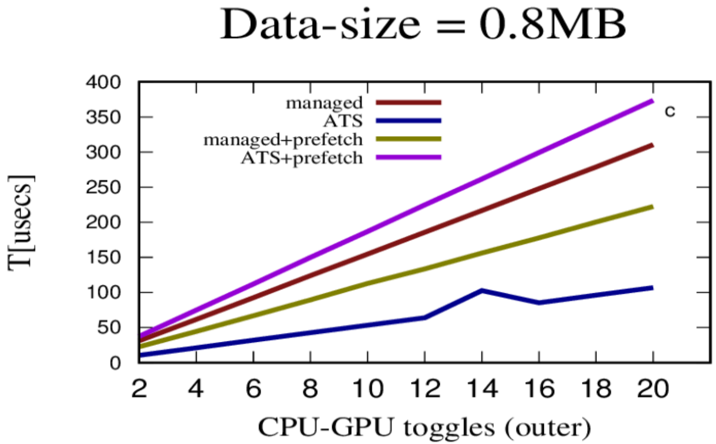
CPU-GPU toggles (outer) = 10 : Consecutive GPU kernel launches (inner) = 2

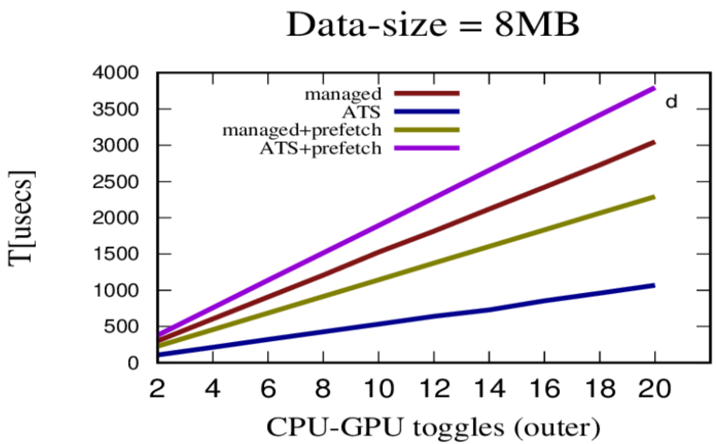
Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	172.5	65.4 (2.6x)	109.7	73.8 (1.4x)

CPU-GPU toggles (inner) = 2 : Consecutive GPU kernel launches (inner) = 10

Data Size	Managed	Managed+prefetch	ATS	ATS+prefetch
0.8MB	55.3	34.5 (1.6x)	126.5	37.13 (3.4x)

TouchOnCPU (Consecutive GPU kernel launches = 2)



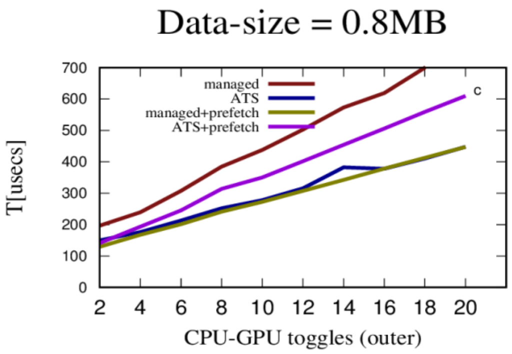


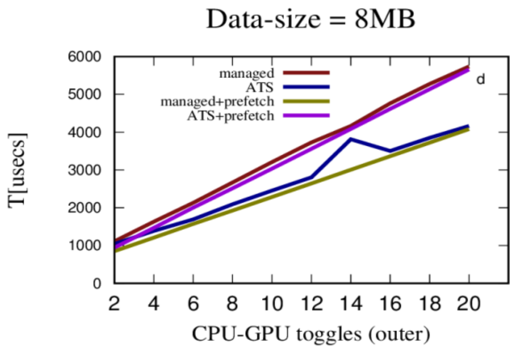
- ATS without prefetch, expectedly is always fastest on CPU
- ATS with prefetch is slowest due to low bandwidth for prefetch on ATS

- Managed benefits with prefetch on both CPU and GPU

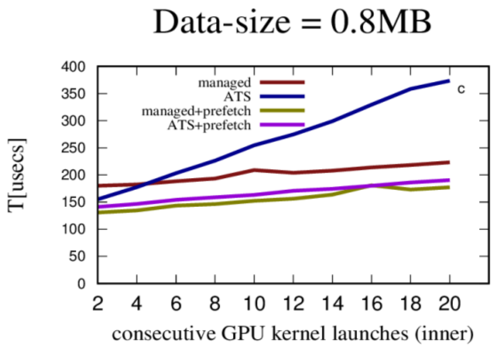
Total time (CPU+GPU)

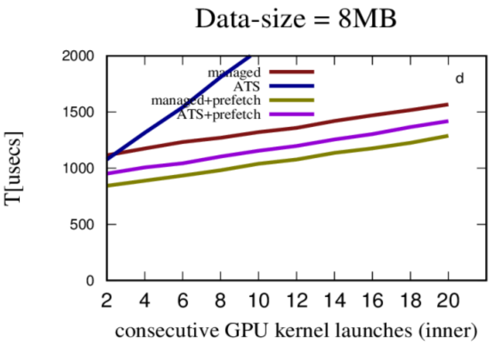
Consecutive kernel launches (inner) = 2





CPU-GPU toggles (outer) = 2





- With increasing data sizes **managed+prefetch is the clear winner.**
- ATS without prefetch gets worse with increasing data sizes for higher number of consecutive GPU kernels but second best if data is utilized more on CPU.
- Managed benefits from prefetch both on CPU and GPU whereas ATS only benefits with prefetch on GPU.

Conclusion

- 4 UVM strategies explored : ATS, managed, ATS+prefetch, managed+prefetch
- Prefetch calls are important to gain performance benefits for GPU kernels.
  - **Usage of prefetch defeats the purpose of UVM.**
- ATS is beneficial only in very few cases compared to managed memory.
  - **The benefits of ATS can be overcome with prefetch directives.**
- Prefetch directives are beneficial for both CPU and GPU kernels for managed memory.
- Prefetch directives with ATS only help GPU kernels.
- When provided with the prefetch directive managed+prefetch was the most successful memory management technique.

Additional Slides

DAXPY (y += a*x)

```
x,y = pointers to an array of double's
a - constant
N rows and M columns
void daxpy(double *x, double *y)
{
  int i,j;
  for(i = 0; i < N; ++i)
    for(j = 0; j < M; ++j)
      y(i,j) += a*x(i,j);
}
```

```
#define y(i,j) = y[i*M+j]

#define x(i,j) = x[i*M+j]
```

DAXPY (Memory Allocation)

```
x,y = pointers to an array of double's
a - constant
N rows and M columns
void daxpy(double *x, double *y)
{
  int i,j;
  for(i = 0; i < N; ++i)
    for(j = 0; j < M; ++j)
      y(i,j) += a*x(i,j);
}
```

```
#if managed_memory
cudaMallocManaged(&x,N*M*sizeof(double));
cudaMallocManaged(&x,N*M*sizeof(double));
#elif defined(ATS)
x = (double*) malloc(N*M*sizeof(double));
y = (double*) malloc(N*M*sizeof(double));
#endif
```

DAXPY - GPU kernel

CPU

```
void daxpyl(double *x, double *y)
{
  int i,j;
  for(i = 0; i < N; ++i)
    for(j = 0; j < M; ++j)
      y(i,j) += a*x(i,j);
}
```

GPU

```
void daxpy_kernel(double *x, double *y)
{
  int i,j;
  for(i = blockIdx.x; i< N; i += gridDim.x)
    for(j = threadIdx.x; j < M;j += blockDim.x)
      y(i,j) += a*x(i,j);
}
```

CPU kernel (TouchOnCPU)

CPU

```
void daxpy_kernel(double *x, double *y)
{
  int i,j;
  for(i = 0; i < N; ++i)
    for(j = 0; j < M; ++j)
      y(i,j) += a*x(i,j);
}
```

GPU

```
void daxpy_kernel(double *x, double *y)
{
  int i,j;
  for(i = blockIdx.x; i< N; i += gridDim.x)
    for(j = threadIdx.x; j < M;j += blockDim.x)
      y(i,j) += a*x(i,j);
}
```

```
void TouchOnCPU(double *x, double *y)
{
  int i,j;
  for(i = 0; i < N; ++i)
    for(j = 0; j < M; ++j)
      y(i,j) -= 0.5;
}
```

Experiment

```
for(outer)//GPU-CPU toggle
{
  for(inner)//consecutive GPU kernel launches
  {
    daxpy_kernel<<<N,32>>>(x,y);
  } //end inner
  TouchOnCPU(y);
}//end outer
```

Experiment

```
for(outer)//GPU-CPU toggle
{
  for(inner)//consecutive GPU kernel launches
  {
    daxpy_kernel<<<N,32>>>(x,y);
  } //end inner
  TouchOnCPU(y);
}//end outer
```

- N = 80
  - Number of SMs in V100
- M = data processed by each threadblock
  - M*sizeof(double)
- outer = times the data is brought back to CPU
- inner = times DAXPY is consecutively launched

Usage of Prefetch directive

```
cudaMemPrefetchAsync(x, N*M*sizeof(double),gpuDeviceId,0);
for(outer)
{
  cudaMemPrefetchAsync(y, N*M*sizeof(double),gpuDeviceId,0);
  for(inner)
  {
    daxpy_kernel<<<N,32>>>(x,y);
  } //end inner
  cudaMemPrefetchAsync(y, N*M*sizeof(double),cudaCPUDeviceId,0);
  TouchOnCPU(y);
}//end outer
```